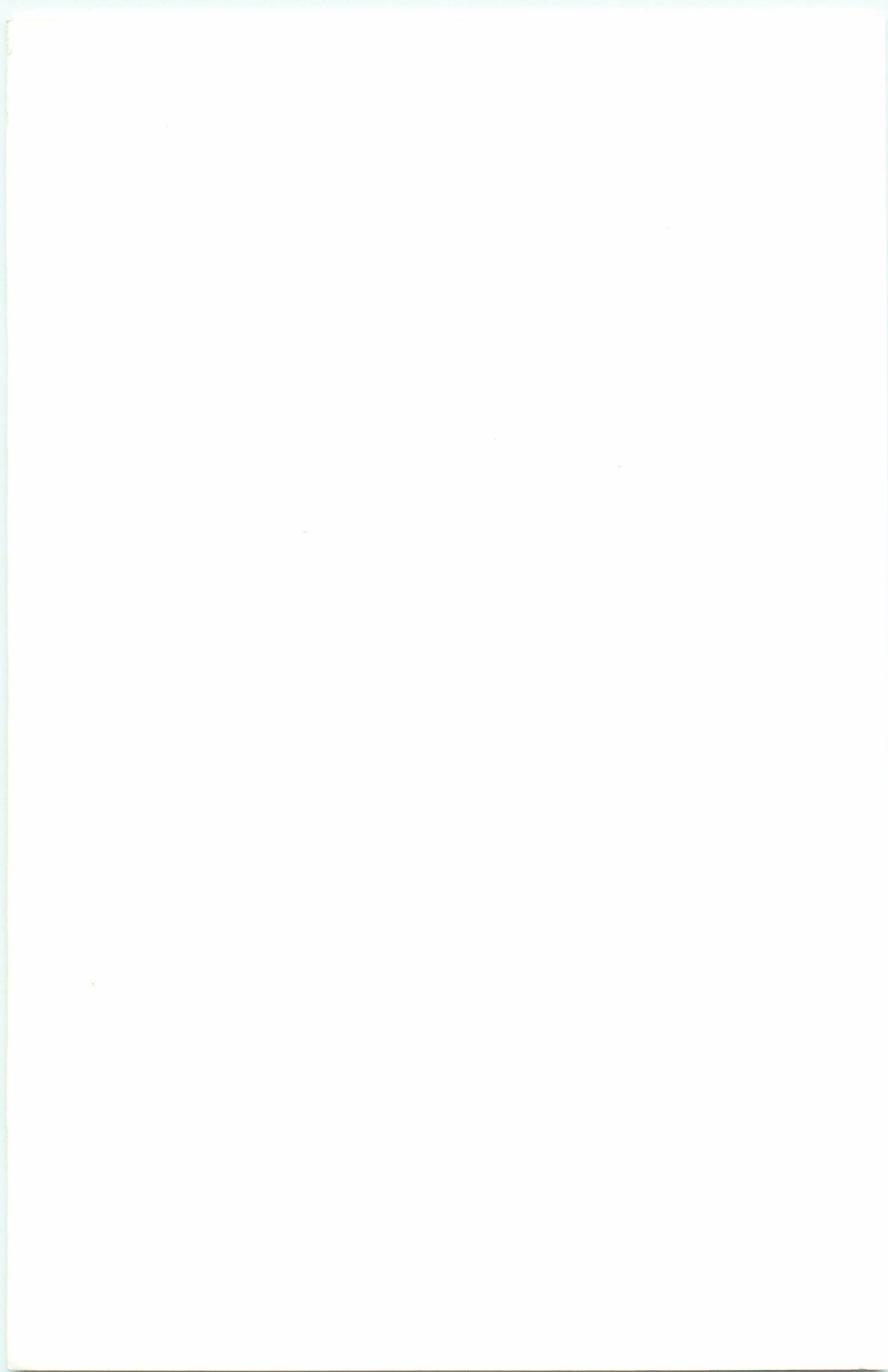
ユーザーズ・マニュアル

MOTOROLA

ENHANCED 32-BIT
MICROPROCESSOR

ユーザーズ・マニュアル

MC68030

ユーザーズ・マニュアル

MC68030

日ごろよりモトローラ製品をご愛顧いただき、誠にありがとうございます。
さて、このたび「MC68030ユーザーズ・マニュアル(第1版)」をお届けいたします。
本書には、MC68030 32ビット第2世代高性能マイクロプロセッサの機能、動作およびプログラミングについて記載されています。
本書がみなさまの一助となることを心から願っております。

1990年12月
日本モトローラ株式会社
半導体事業部

目　　次

注

本書では、信号を強制的に特定の状態にすることを指す「アサート」および「ネゲート」という用語を使用します。特に、「アサーション」および「アサート」はアクティブまたは真である信号を指し、「ネゲーション」および「ネゲート」は非アクティブまたは偽である信号を指します。これらの用語は、それらが表わす電圧レベル（“H” または “L”）とは無関係に使用しています。

本書は読者として、システム設計者、システム・プログラマ、およびアプリケーション・プログラマを対象にしたものです。システム設計者は、特に第1、5、6、7、13、および14章、そして付録Aを重点的に、すべての章に関してある程度の知識が必要です。システムにコプロセッサをインプリメントする設計者は、第10章を完全に理解しておく必要があります。また、システム・プログラマは第1、2、3、4、6、8、9、および11章、そして付録Aを理解しておいてください。アプリケーション・プログラマに必要な情報の大部分は、第1、2、3、4、9、11、および12章、そして付録Aに記載してあります。

また、本書の読者には他のM68000ファミリのユーザや、これらのマイクロプロセッサを熟知していない方がいます。他のファミリのユーザに対し、本書の随所に他のモトローラのマイクロプロセッサとの類似点や違いに関する参照の指示を載せています。ただし、第1章および付録Aでは、MC68030をファミリの他の製品と対比させて、その特長を列挙するとともに相違点を説明しています。

第 1 章
概　要

MC68030はモトローラの第2世代の完全32ビット高性能マイクロプロセッサです。MC68030はM68000ファミリ・デバイスのメンバで、中央処理ユニット(CPU)コア、データ・キャッシュ、命令キャッシュ、高性能バス・コントローラ、およびメモリ管理ユニットを、シングルVLSIデバイスに搭載しています。プロセッサは20MHz以上のクロック・スピードで動作するよう設計されています。MC68030は、32ビット・レジスタおよびデータ・パス、32ビット・アドレス、豊富な命令セット、および柔軟なアドレッシング・モードをインプリメントしています。

MC68030はM68000ファミリの初期のメンバとオブジェクト・コード・レベルで上位互換性があり、オンチップ・メモリ管理ユニット、データ・キャッシュ、および改善が加えられたバス・インタフェースなど、さらに多彩な機能を追加しています。MC68020で導入された柔軟なコプロセッサ・インタフェースを継承し、それを通してMC68881またはMC68882浮動小数点演算コプロセッサによる完全なIEEE浮動小数点サポートを提供します。また、このマイクロプロセッサの内部の機能ブロックは並列に動作するよう設計されており、命令の実行をオーバラップさせることができます。命令の実行だけでなく、内部キャッシュ、オンチップ・メモリ管理ユニット、および外部バス・コントローラもすべて並列に動作します。

MC68030は32ビット・アドレスおよび32ビット・データにより、MC68020の非多重化バス構造を完全にサポートします。MC68030バスには非同期および同期バス・サイクルおよびバースト・データ転送をサポートする高性能コントローラがあります。また、プロセッサが外部デバイスとの間でオペランドを転送するごとにサイクル単位で自動的にデバイスのポート・サイズを決定する、MC68020ダイナミック・バス・サイジング機構もサポートしています。

MC68030のブロック図を図1-1に示します。プロセッサが要求する命令およびデータは、可能なときは常に内部キャッシュから供給されます。メモリ管理ユニット(MMU)は、アドレス変換キャッシュ(ATC)を利用することにより、プロセッサが生成した論理アドレスを物理アドレスに変換します。バス・コントローラは、物理アドレスにおけるCPUとメモリまたはデバイス間のデータ転送を管理します。

1.1　特　長

MC68030マイクロプロセッサの特長は以下のとおりです。

- MC68020および初期のM68000マイクロプロセッサとオブジェクト・コード・コンパチブル
- 完全32ビット非多重化アドレスおよびデータ・バス
- 16個の32ビット汎用データおよびアドレス・レジスタ
- 2個の32ビット・スーパバイザ・スタック・ポインタおよび10個の特殊制御レジスタ

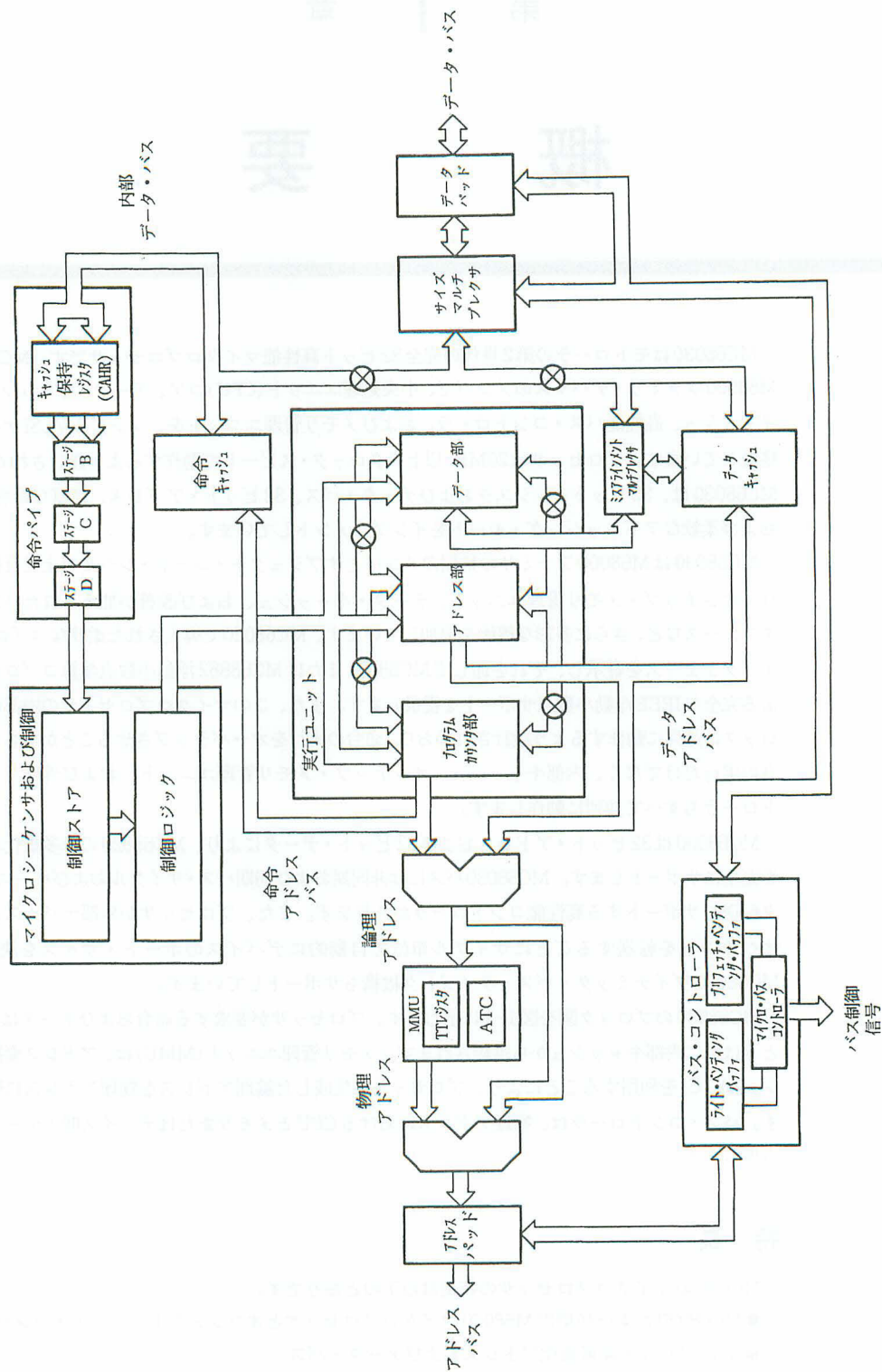



図1-1 MC68030 ブロック図

- 同時アクセス可能な256バイト命令キャッシュおよび256バイト・データ・キャッシュ
- 命令実行および内蔵キャッシュ・アクセスと並行してアドレス変換を行なうページ方式のメモリ管理ユニット
- 2つの透過セグメントにより、定義済み物理アドレス間でデータ・ブロックを転送するグラフィックス・アプリケーションなどのシステムに対し、物理メモリへの未変換アクセスの定義が可能
- 高度な並列処理機能をもつパイプライン・アーキテクチャにより、バス転送および命令実行とオーバラップして、内部キャッシュへのアクセスが可能
- 高性能バス・コントローラで、すべての物理アドレス空間に対して、非同期バス・サイクル(最小3クロック)、同期バス・サイクル(最小2クロック)、およびバースト・データ転送(最小1クロック)をサポート
- ダイナミック・バス・サイジングで8、16、32ビット・メモリおよび周辺デバイスをサポート
- M68000コプロセッサ・インタフェース付きのコプロセッサのサポート。MC68881/MC68882浮動小数点演算コプロセッサにより完全なIEEE浮動小数点をサポート
- 4ギガバイトの論理および物理アドレッシング範囲
- モトローラのHCMOSテクノロジでインプリメントされており、CMOSとHMOS(高密度NMOS)ゲートを最大速度、低消費電力、および最適なダイ・サイズで結合
- 20MHz以上のプロセッサ・スピード

MMUとデータおよび命令キャッシュをインプリメントしたため、性能と機能性が向上しています。また、高性能バス・コントローラおよび内部並列処理によっても、システム性能の向上が図られています。さらに、バス・インタフェースの改善、物理的サイズの縮小、および消費電力の低減などの相乗効果によって、システム・コストが下がるため、コスト・パフォーマンスの目標を達成することができます。

1.2　M68000ファミリへのMC68030の拡張

MC68020のオンチップ命令キャッシュに加えて、MC68030は内部データ・キャッシュを備えています。リード・サイクル時にアクセスされるデータは内蔵キャッシュに記憶することができ、次回のアクセスで利用できます。データ・キャッシュは、命令が必要とするデータ・オペランドがすでにデータ・キャッシュに存在しているときに外部バス・サイクル数を減少させます。

内蔵キャッシュは内部で単一クロック・サイクルでアクセス可能なため、一段と性能が向上します。さらに、バス・コントローラは2クロック・サイクル同期モードおよびデータをロング・ワード当たり1クロックで転送可能なバースト・モード・アクセスを提供します。

MC68030高性能マイクロプロセッサは、オンチップ・メモリ管理ユニットを内蔵しており、これによってCPUコア、内部キャッシュ、およびバス・コントローラと並行してアドレス変換処理を実行することができます。

このほか、エミュレーションやシステム解析をサポートする信号も用意されています。外部デバッグ装置は、ブレークポイント処理時にオンチップ・キャッシュおよびMMUをディセーブルすることによって、MC68030の内部状態を凍結させることができます。さらに、MC68030が次のような情報を表示します。

1. 命令パイプのリフィルの先頭
2. 命令の境界
3. ペンディングのトレースまたは割込み処理
4. 例外処理
5. ホルト条件

このステータスおよび制御情報により、外部デバッグ装置はMC68030の動作をトレースし、MC68030との間で独占的な対話処理が可能なため、システムのデバッグ作業を効率よく低減できます。

1.3 プログラミング・モデル

MC68030のプログラミング・モデルは、ユーザ・モデルとスーパバイザ・モデルの2つのグループのレジスタで構成されます。これはユーザおよびスーパバイザ特権レベルに対応しています。ユーザ特権レベルで実行されるユーザ・プログラムは、ユーザ・モデルのレジスタしか使用できません。スーパバイザ・レベルで実行されるシステム・ソフトウェアは、スーパバイザ・レベルの制御レジスタを使用してスーパバイザ機能を実行します。

図1-2にユーザ・プログラミング・モデルを示します。これは、次のような16個の32ビット汎用レジスタおよび2個の制御レジスタで構成されています。

- 汎用32ビット・レジスタ(D0～D7、A0～A7)
- 32ビット・プログラム・カウンタ(PC)
- 8ビット・コンディション・コード・レジスタ(CCR)

スーパバイザ・プログラミング・モデルは、ユーザが利用できるレジスタと14個の制御レジスタで構成されます。

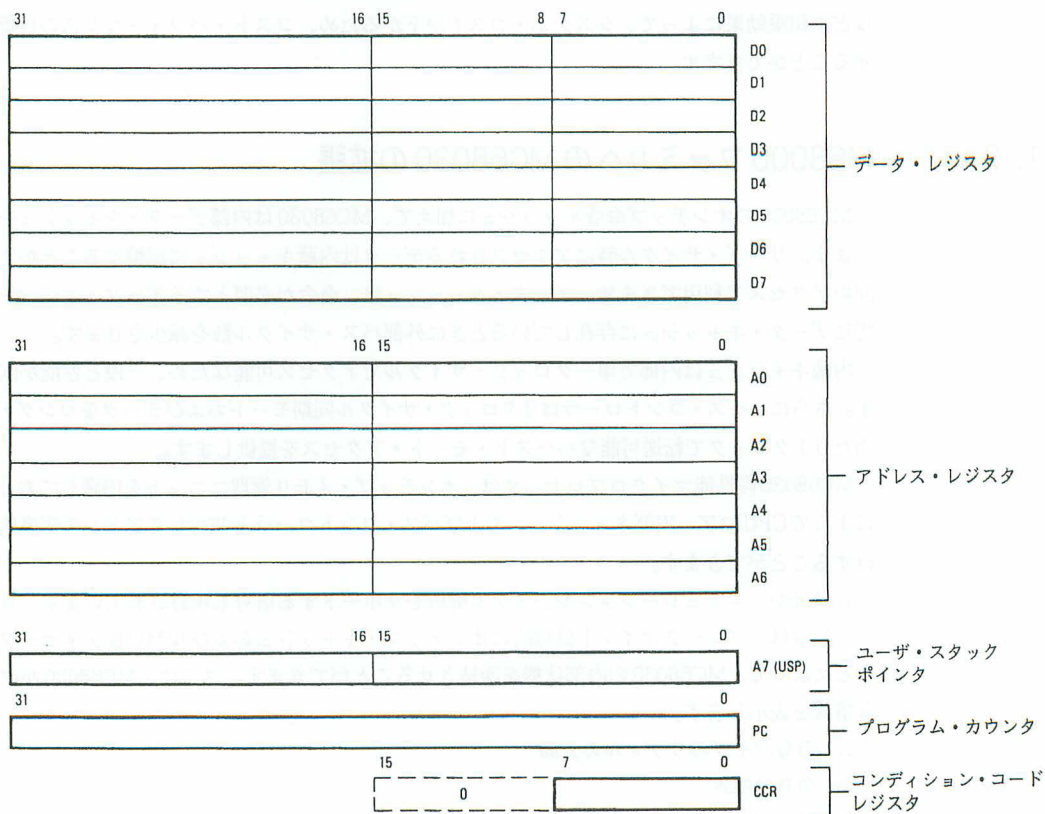



図1-2 ユーザ・プログラミング・モデル

- 2個の32ビット・スーパバイザ・スタック・ポインタ(ISPおよびMSP)
- 16ビット・ステータス・レジスタ(SR)
- 32ビット・ベクタ・ベース・レジスタ(VBR)
- 32ビット・オルタネート・ファンクション・コード・レジスタ(SFCおよびDFC)
- 32ビット・キャッシュ制御レジスタ(CACR)
- 32ビット・キャッシュ・アドレス・レジスタ(CAAR)
- 64ビットCPUルート・ポインタ(CRP)
- 64ビット・スーパバイザ・ルート・ポインタ(SRP)
- 32ビット変換制御レジスタ(TC)
- 32ビット透過変換レジスタ(TT0およびTT1)
- 16ビットMMUステータス・レジスタ(MMUSR)

ユーザ・プログラミング・モデルは、以前のM68000ファミリ・マイクロプロセッサから変わっていません。スーパバイザ・プログラミング・モデルは、ユーザ・プログラミング・モデルを補完するもので、応答性にすぐれたオペレーティング・システム機能、I/O制御およびメモリ管理サブシステムを実現するためにスーパバイザ特権レベルを利用するMC68030システム・プログラマだけが使用します。このスーパバイザ・プログラミング・モデルには、MC68030の特別な機能をアクセスおよびイネーブルするためのすべての制御機能があります。この分離は十分配慮して実現されていますので、すべてのアプリケーション・ソフトウェアは非特権ユーザ・レベルで動作するように書かれ、任意のM68000プラットホームから修正なしでMC68030に移植できるようになっています。通常、システム・ソフトウェアは新しい設計に移行する際にシステム・プログラマが修正を加えるため、制御機能はスーパバイザ・プログラミング・モデルに適切に配置されています。たとえば、MC68030の透過変換機能および2個の変換レジスタはMC68030のファミリ・スーパバイザ・プログラミング・モデルに新しく追加されたものです。スーパバイザ・コードだけがこの機能を使用し、ユーザ・アプリケーション・プログラムは影響を受けません。

レジスタD0〜D7はビットおよびビット・フィールド（1から32ビット)、バイト(8ビット)、ワード(16ビット)、ロング・ワード(32ビット)、およびクワッド・ワード(64ビット)操作のためのデータ・レジスタとして使用します。レジスタA0〜A6、ユーザ・スタック・ポインタ、割込みスタック・ポインタ、およびマスタ・スタック・ポインタは、ソフトウェア・スタック・ポインタまたはベース・アドレス・レジスタとして使用できるアドレス・レジスタです。レジスタA7（図1-3でA7′およびA7″として示しています)は、ユーザ特権レベルではユーザ・スタック・ポインタに、またスーパバイザ特権レベルでは割込みまたはマスタ・スタック・ポインタのいずれかに適用されるレジスタ名です。スーパバイザ特権レベルでは、アクティブなスタック・ポインタ(割込みまたはマスタ)をスーパバイザ・スタック・ポインタ(SSP)とよびます。さらに、アドレス・レジスタをワードおよびロング・ワード操作に使用することができます。16個の汎用レジスタ(D0〜D7、A0〜A7)は、すべてインデックス・レジスタとして使用できます。

プログラム・カウンタ(PC)は、MC68030が次に実行する命令のアドレスを保持します。命令実行中および例外処理中、プロセッサはPCの内容を自動的にインクリメントするか、または適宜新しい値をPCに入れます。

ステータス・レジスタSR(図1-4)は、プロセッサの状態を記憶します。これは前の処理の結果を反映するコンディション・コードを内容とし、プログラムの条件付き命令を実行するために使用できます。コンディション・コードは拡張(X)、ネガティブ(N)、ゼロ(Z)、オーバフロー(V)、およびキャリ(C)です。コンディション・コードを含むユーザ・バイトは、ユーザ特権レベルで利用可能な唯一のステータス・レジスタ情報であり、ユーザ・プログラムでCCRとして参照されます。スーパバイザ特権レベルでは、ソフトウェアは割込み優先順位マスク(3ビット)および追加制御ビットを含

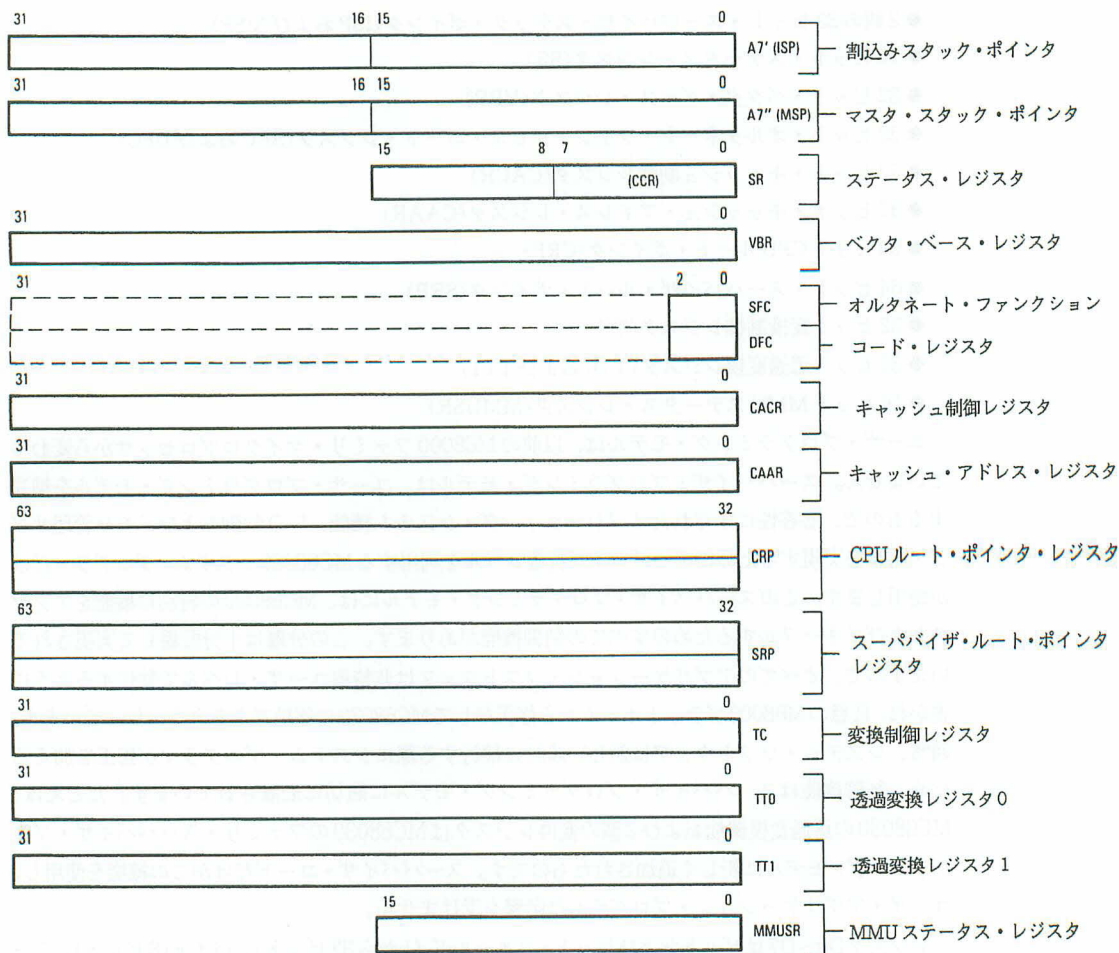



図1-3　スーパバイザ・プログラミング・モデル追補

むステータス・レジスタ全体にアクセスできます。これらのビットは、プロセッサが次の状態にあるかどうかを示します。

1. 2つのトレース・モード(T1、T0)のいずれか
2. スーパバイザまたはユーザ特権レベル(S)
3. マスタまたは割込みモード(M)

ベクタ・ベース・レジスタ(VBR)には、メモリ内の例外ベクタ・テーブルのベース・アドレスがあります。ベクタ・テーブルにアクセスするために、例外ベクタのディスプレースメントがこのレジスタの値に加算されます。

オルタネート・ファンクション・コード・レジスタSFCおよびDFCには、3ビットのファンクション・コードがあります。ファンクション・コードは、オプションにより8つの4ギガバイト・アドレス空間を提供する、32ビットのリニア・アドレスの拡張と考えることができます。ファンクション・コードはプロセッサによって自動的に生成され、ユーザおよびスーパバイザ特権レベルにあるデータおよびプログラムのためのアドレス空間、およびプロセッサ機能(たとえばコプロセッサ通信など)に使用するCPUアドレス空間を選択します。レジスタSFCおよびDFCは、特定の命令によって使用され、操作のためのファンクション・コードを明示的に指定します。

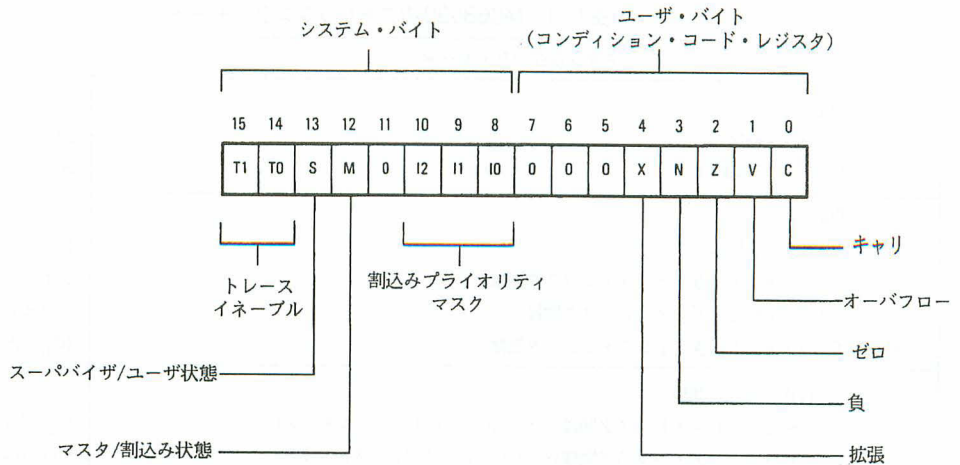



図1-4　ステータス・レジスタ

キャッシュ制御レジスタ(CACR)は、MC68030のオンチップ命令およびデータ・キャッシュを制御します。キャッシュ・アドレス・レジスタ(CAAR)はキャッシュ制御機能のアドレスを記憶します。

CPUルート・ポインタ(CRP)は、MC68030が現在実行しているタスクに対する変換ツリーのルートを指すポインタを保持しています。このツリーには、タスクのアドレス空間のマッピング情報があります。スーパバイザ・ルーチンに個別のアドレス空間を与えるようにMC68030を構成すると、スーパバイザ・ルート・ポインタ(SRP)は、スーパバイザのアドレス空間を記述する変換ツリーのルートを指すポインタをもっています。

変換制御レジスタ(TC)は、アドレス変換を制御する複数のフィールドで構成されます。これらのフィールドは、アドレス変換のイネーブルおよびディセーブル、スーパバイザ・アドレス空間のためのSRP使用のイネーブルおよびディセーブルを行ない、さらにアドレスを変換する際にファンクション・コードを選択するか、あるいは無視します。その他のフィールドはメモリ・ページのサイズ、変換で使用するアドレス・ビット数、および変換テーブルの構造を定義します。

透過変換レジスタTT0およびTT1により、メモリの個別ブロックをアドレス変換を行なわないで直接アクセス可能なように指定することができます。これらの領域の論理アドレスは、メモリ・アクセス時の物理アドレスになります。ファンクション・コードおよびアドレスの最上位8ビットは、メモリの領域とアクセスのタイプを定義するのに使用でき、リードまたはライトのいずれか、または両方のタイプのメモリ・アクセスを直接マッピングできます。透過変換機能により、メモリまたはI/O空間において、オンチップ・アドレス変換キャッシュのコンテクストを妨害したり、変換テーブルのルックアップに付随した遅延を生じることなく、大きなデータ・ブロックを高速で移動することができます。この機能はグラフィックス、コントローラ、およびリアルタイム・アプリケーションに役立ちます。

MMUステータス・レジスタ(MMUSR)は、アドレス変換キャッシュまたは変換ツリーで、ある論理アドレスをサーチした結果生じる、メモリ管理ステータス情報を保持します。

1.4　データ・タイプおよびアドレッシング・モード

次の7種類の基本データ・タイプがサポートされています。

- ビット
- ビット・フィールド(連続ビット・フィールド、長さ1 ～32ビット)

表1-1　MC68030のアドレッシング・モード

アドレッシング・モード	シンタックス
レジスタ直接	
データ・レジスタ直接	Dn
アドレス・レジスタ直接	An
レジスタ間接	
アドレス・レジスタ間接	(An)
ポストインクリメント付きアドレス・レジスタ間接	(An) +
プリデクリメント付きアドレス・レジスタ間接	− (An)
ディスプレースメント付きアドレス・レジスタ間接	(d_{16}, An)
インデックス付きレジスタ間接	
インデックス付きアドレス・レジスタ間接(8ビット・ディスプレースメント)	(d_8, An, Xn)
インデックス付きアドレス・レジスタ間接(ベース・ディスプレースメント)	(bd, An, Xn)
メモリ間接	
ポストインデックス付きメモリ間接	([bd, An], Xn, od)
プリインデックス付きメモリ間接	([bd, An, Xn], od)
ディスプレースメント付きプログラム・カウンタ間接	(d_{16}, PC)
インデックス付きプログラム・カウンタ間接	
インデックス付きPC間接(8ビット・ディスプレースメント)	(d_8, PC, Xn)
インデックス付きPC間接(ベース・ディスプレースメント)	(bd, PC, Xn)
プログラム・カウンタ・メモリ間接	
ポストインデックス付きPCメモリ間接	([bd, PC], Xn, od)
プリインデックス付きPCメモリ間接	([bd,PC, Xn], od)
絶対	
絶対ショート	(xxx).W
絶対ロング	(xxx).L
イミディエイト	# 〈data〉

注：Dn　= データ・レジスタ、D0〜D7

An　= アドレス・レジスタ、A0〜A7

d_8, d_{16} = 2の補数、または符号拡張ディスプレースメント。実効アドレス計算の一部として加算される。サイズは8(d_8)または16(d_{16})ビット。省略した場合、アセンブラは0の値を使用する。

Xn　= インデックス・レジスタとして使用するアドレス・レジスタまたはデータ・レジスタ。形式はXn.SIZE＊SCALEで、SIZEは.Wまたは.L(インデックス・レジスタのサイズを示す)で、SCALEは1、2、4または8(インデックス・レジスタにSCALEが乗算される)。SIZEおよびSCALE、またはその一方の使用はオプション。

bd　= 2の補数ベース・ディスプレースメント。これが存在する場合、サイズは16または32ビット。

od　= 外側ディスプレースメント。メモリ間接の後に実効アドレス計算の一部として加算される。この使用はオプションで、16または32ビットのサイズが指定できる。

PC　= プログラム・カウンタ

〈data〉 = 8, 16, または32ビットのイミディエイト値

( )　=実効アドレス

[ ]　=ロング・ワード・アドレスに対する間接アドレスとして使用

表1-2 命令セット

ニーモニック	内　容
ABCD	拡張付き10進加算
ADD	加算
ADDA	アドレス加算
ADDI	イミディエイト加算
ADDQ	クイック加算
ADDX	拡張加算
AND	論理積
ANDI	イミディエイト論理積
ASL,ASR	算術右または左シフト
Bcc	条件分岐
BCHG	ビット・テストと変更
BCLR	ビット・テストとクリア
BFCHG	ビット・フィールドのテストと変更
BFCLR	ビット・フィールドのテストとクリア
BFEXTS	符号付きビット・フィールド抽出
BFEXTU	符号なしビット・フィールド抽出
BFFFO	ビット・フィールド内の最初の1検出
BFINS	ビット・フィールド挿入
BFSET	ビット・フィールドのテストとセット
BFTST	ビット・フィールドのテスト
BKPT	ブレーク・ポイント
BRA	無条件分岐
BSET	ビット・テストとセット
BSR	サブルーチン分岐
BTST	ビット・テスト
CAS	オペランドの比較と交換
CAS2	デュアル・オペランドの比較と交換
CHK	レジスタ境界チェック
CHK2	レジスタの上限および下限のチェック
CLR	クリア
CMP	比較
CMPA	アドレス比較
CMPI	イミディエイト比較
CMPM	メモリ比較
CMP2	レジスタ上限および下限の比較
DBcc	条件テスト・デクリメントおよび分岐
DIVS, DIVSL	符号付き除算
DIVU, DIVUL	符号なし除算
EOR	排他的論理和
EORI	イミディエイト排他的論理和
EXG	レジスタの交換
EXT, EXTB	符号拡張
ILLEGAL	不当命令トラップ処理要求
JMP	ジャンプ
JSR	サブルーチンへのジャンプ
LEA	実効アドレスのロード
LINK	リンクと割付け
LSL, LSR	論理右および左シフト
MOVE	データ転送
MOVEA	アドレス転送
MOVE CCR	コンディション・コード・レジスタの転送
MOVE SR	ステータス・レジスタの転送

ニーモニック	内　容
MOVE USP	ユーザ・スタック・ポインタの転送
MOVEC	制御レジスタ転送
MOVEM	複数レジスタ転送
MOVEP	周辺データ転送
MOVEQ	クイック転送
MOVES	オルタネート・アドレス空間転送
MULS	符号付き乗算
MULU	符号なし乗算
NBCD	拡張付き10進ネゲート
NEG	ネゲート
NEGX	拡張付きネゲート
NOP	無操作
NOT	論理否定
OR	論理和
ORI	イミディエイト論理和
ORI CCR	ｺﾝﾃﾞｨｼｮﾝ･ｺｰﾄﾞとのイミディエイト論理和
ORI SR	ｽﾃｰﾀｽ･ﾚｼﾞｽﾀとのイミディエイト論理和
PACK	パックBCD
PEA	実効アドレスのプッシュ
PFLUSH	ATCのエントリのフラッシュ
PFLUSHA	ATCの全エントリのフラッシュ
PLOADR, PLOADW	ATCへのエントリのロード
PMOVE	MMUレジスタ転送
PMOVEFD	ラッシュ・ディセーブル付きMMUレジスタ転送
PTESTR, PTESTW	論理アドレスのテスト
RESET	外部デバイスのリセット
ROL, ROR	右または左ローテイト
ROXL, ROXR	拡張付き右または左ローテイト
RTD	リターンおよびパラメータの割当て解除
RTE	例外処理からのリターン
RTR	リターンおよびコンディション・コードの回復
RTS	サブ・ルーチンからのリターン
SBCD	拡張付き10進減算
Scc	条件セット
STOP	ストップ
SUB	減算
SUBA	アドレス減算
SUBI	イミディエイト減算
SUBQ	クイック減算
SUBX	拡張付き減算
SWAP	レジスタ・ワードの交換
TAS	オペランドのテストとセット
TRAP	トラップ
TRAPcc	条件トラップ
TRAPV	オーバフロー・トラップ
TST	オペランドのテスト
UNLK	リンク解除
UNPK	アンパックBCD

コプロセッサ命令

ニーモニック	内　容
cpBcc	コプロセッサ条件分岐
cpDBcc	コプロセッサ条件テスト、デクリメントおよび分岐
cpGEN	コプロセッサの一般機能

ニーモニック	内　容
cpRESTORE	コプロセッサ・リストア機能
cpSAVE	コプロセッサ・セーブ機能
cpScc	コプロセッサ条件セット
cpTRAPcc	コプロセッサ条件トラップ

- BCD ディジット(パック型式：2ディジット/バイト、アンパック型式：1ディジット/バイト)
- バイト整数(8ビット)
- ワード整数(16ビット)
- ロング・ワード整数(32ビット)
- クワッド・ワード整数(64ビット)

このほか、メモリ・アドレスなどのデータ・タイプもサポートされています。また、コプロセッサ機構によって、MC68881およびMC68882浮動小数点演算コプロセッサによる浮動小数点演算が、特殊なユーザ定義のデータ・タイプおよび機能と同様に直接サポートされます。

表1-1 に示すように、18種類のアドレッシング・モードがあり、そのうち以下の9種類が基本タイプとなっています。

- レジスタ直接
- レジスタ間接
- インデックス付きレジスタ間接
- メモリ間接
- ディスプレースメント付きプログラム・カウンタ間接
- インデックス付きプログラム・カウンタ間接
- プログラム・カウンタ・メモリ間接
- 絶対
- イミディエイト

レジスタ間接アドレッシング・モードでは、ポストインクリメント、プリデクリメント、オフセット、およびインデックスを指定することができます。プログラム・カウンタ相対モードでもインデックスおよびオフセットを指定できます。MC68020と同様に、この2つのモードはメモリを使用して間接参照が可能なように拡張されています。これらのアドレッシング・モードのほかにも、多くの命令でコンディション・コード・レジスタ、スタック・ポインタ、およびプログラム・カウンタのいずれか、あるいは全部の使用を指定することができます。

1.5 命令セットの概要

MC68030 の命令セットを表1-2 に示します。構造化高級言語や高度なオペレーティング・システムをサポートするために、命令セットが強化されています。多くの命令はバイト、ワード、またはロング・ワードを扱い、ほとんどの命令が18種類のアドレッシング・モードを任意に使用することができます。

1.6 仮想メモリ/マシンの概念

MC68030 の全アドレッシング範囲は、8つのアドレス空間のそれぞれで4ギガバイト(4, 294, 967, 296バイト)です。たいていのMC68030システムはこれより小さい物理メモリを実装していますが、仮想メモリ手法を用いて、各ユーザ・プログラムに対し4ギガバイトのメモリがフルに存在するように見えるシステムにすることができます。

仮想メモリ・システムでは、実際にシステムに実装されている物理メモリが小さくても、あたかも大容量のメモリが存在するかのようにユーザ・プログラムを書くことができます。同様に、テープ・ドライブ、ディスク・ドライブ、プリンタ、ターミナルなど、システムに実際には存在しない装置に対し、ユーザ・プログラムでアクセスできるようにシステムを設計することができます。ソフトウェアで適切にエミュレートすれば、物理システムをユーザ・プログラムにとって別のM68000

コンピュータ・システムに見えるようにすることができ、エミュレートされたシステムの全資源に完全にアクセスできます。このようにエミュレートされたシステムを仮想マシンとよんでいます。

1.6.1　仮想メモリ

仮想メモリをサポートするシステムは、プロセッサから直接アクセスできる小容量の高速物理メモリを用意し、それよりもはるかに大きい“仮想”メモリ・イメージを大容量ディスク・ドライブなどの補助記憶装置で保持します。プロセッサが仮想メモリ・マップ上のあるロケーションにアクセスしようとしたときに、そのロケーションの情報が物理メモリ上に存在しなかった場合は、“ページ・フォールト”が発生します。ページ・フォールトが発生すると、そのロケーションへのアクセスは一時的に中断され、必要なデータが補助記憶装置からフェッチされて物理メモリ上に置かれます。その後、中断されていたアクセスが再開または継続されます。

MC68030 は命令継続法を使って仮想メモリをサポートします。バス・サイクルがバス・エラー信号によってターミネートされると、マイクロプロセッサは現在の命令を中断して仮想メモリ・バス・ハンドラを実行します。バス・エラー・ハンドラは実行を完了すると、エラーが検出されていたときに実行中であったプログラムに制御を返し、フォールトが発生したバス・サイクルを再実行(必要な場合)した後、中断されていた命令を継続して実行させます。

1.6.2　仮想マシン

仮想マシン・システムの代表的な使用例として、開発中でありまだプログラム作成に利用できない新しいマシンに対する、オペレーティング・システムなどのソフトウェアの開発があります。仮想マシン・システムでは、管理OSが新しいマシンのハードウェアをエミュレートし、新しいソフトウェアをあたかも新しいハードウェア上で稼働しているかのように実行し、デバッグできるようにします。新しいソフトウェアは、管理OSによって管理され、管理OSより低い特権レベルで実行されます。したがって、新しいソフトウェアが物理的に存在しない(それゆえに、エミュレートする必要がある)仮想資源にアクセスしようとすると、管理OSにトラップされ、管理OSのソフトウェアで処理されます。

MC68030の仮想マシンでは、ユーザ特権レベルで仮想アプリケーションが実行されます。管理OSはスーパバイザ特権レベルで動作し、新OSがスーパバイザ資源へアクセスしようとしたり、特権命令を実行しようとすると、管理OSへのトラップが発生します。

メモリ・マップ入出力方式のシステムでは、仮想入出力デバイスをサポートするのに命令の継続を使用しています。仮想デバイスのコントロール・レジスタおよびデータ・レジスタは、メモリ・マップの中でシミュレートされます。仮想レジスタにアクセスするとフォールトが発生し、ソフトウェアによってそのレジスタの機能がエミュレートされます。

1.7　メモリ管理ユニット

メモリ管理ユニット(MMU)は、メモリに格納されている変換テーブルを使用して、論理アドレスを物理アドレスに変換することによって仮想メモリ・システムをサポートします。MMUは最後に使用したアドレス変換キャッシュ(ATC)にアドレス・マッピングを格納します。ATCにCPUが要求するバス・サイクルのアドレスがあるときには、変換テーブルのサーチは行なわれません。MMUの特長は以下のとおりです。

- テーブル空間の利用効率を高めるショートおよびロング・フォーマット・ディスクリプタによる多重レベル変換テーブル
- マイクロコードによるテーブル・サーチの自動実行

- 22エントリのフル・アソシアティブATC
- アドレス変換、内部命令、およびデータ・キャッシュ・アクセスを並列に実行
- 256～32Kバイトの8ページ・サイズが利用可能
- 2つのオプションのトランスペアレント・ブロック
- ユーザおよびスーパバイザ・ルート・ポインタ・レジスタ
- 書込み保護およびスーパバイザ保護属性
- ソフトウェアによる変換のイネーブル/ディセーブル
- 外部$\overline{\text{MMUDIS}}$信号により変換をディセーブル可能
- テーブルおよびATCにおいて使用および変更ビットを自動的に維持
- キャッシュ・インヒビット出力($\overline{\text{CIOUT}}$)信号をページ単位でアサート可能
- 15の上位アドレス・ビットを無視できる32ビットの内部論理アドレス
- 3ビットのファンクション・コードで別々のアドレス空間をサポート
- 32ビットの物理アドレス

MMUが実行するメモリ管理機能をデマンド・ページ・メモリ管理とよびます。タスクは実行時に必要なメモリ領域を指定するため、メモリの割付けはデマンド・ベースでサポートされます。要求されたメモリへのアクセスが、現在システムでマップされていない場合は、そのアクセスによりオペレーティング・システムに対して必要なメモリ・イメージのロードまたは割付けが要求されます。MC68030では、物理メモリはページ・フレームとよぶ指定バイト数よりなるブロックで管理されているため、ページ・メモリ管理手法を使用しています。論理アドレス空間は、ページ・フレームと同じバイト数をもつ固定サイズ・ページに分割されています。メモリ管理は物理ページ・アドレスを論理ページに割り当てます。次にシステム・ソフトウェアは補助記憶装置とメモリ間で1回に1ページ以上のデータを変換します。

1.8 パイプライン・アーキテクチャ

MC68030は3段のパイプライン内部アーキテクチャを使用して、命令のスループットを最適化します。このパイプラインによって、1つの命令の3ワードまで、あるいは3個までの連続した命令を並行してデコードすることができます。

1.9 キャッシュ・メモリ

参照の局所性のために、プログラムで使用される命令やデータは、短時間のうちに再使用される確率が高くなっています。さらに、現在使用中の命令およびデータの近くにある命令やデータも、短期間のうちに再使用される確率が高いといえます。このような局所性の特質を生かすために、MC68030は2つの論理キャッシュ、1つのデータ・キャッシュ、および1つの命令キャッシュをチップ上に内蔵しています。

各キャッシュは、それぞれが4つのロング・ワード(16バイト)ブロックをもつ16個のエントリよりなる256バイトの情報を記憶します。プロセッサは1回に1つのロング・ワード、あるいはバースト・モード・アクセス中には連続した4つのロング・ワードをキャッシュ・エントリに入れます。バースト動作モードは、キャッシュを効率よく満たすだけでなく、実行タスクの局所性のために近い将来に必要になると思われる近傍の命令やデータ・アイテムをも捕捉します。

キャッシュによって、プロセッサがメモリから情報をフェッチするのに必要なバス・サイクル数が減り、システムの他のバス・マスタが利用できるバス・バンド幅が増えるため、システムの全体性能が向上します。MC68030でデータ・キャッシュが追加されたことにより、キャッシュ手法の恩

恵がメモリ・アクセス全体にまで拡大されました。ライト・サイクル中、データ・キャッシュ回路はデータをメモリ内のアイテムだけでなく、キャッシュに入っているデータ・アイテムにも書き込み、キャッシュ内のデータとメモリ内のデータの一貫性を維持します。しかし、キャッシュにないデータの書込みについては、キャッシュ制御レジスタ(CACR)で選択されているライト・アロケーション方式によって、データ・アイテムがキャッシュに格納される場合とされない場合があります。

第 2 章

データ構成および アドレッシング機能

マイクロプロセッサによるメモリへの外部参照は、ほとんどがプログラム参照かデータ参照のいずれかです。それらは命令ワードまたは命令のオペランド(データ・アイテム)のいずれかをアクセスします。プログラム参照は、プログラム空間、プログラム命令をもつメモリ・セクション、および命令ストリームにある任意のイミディエイト・データ・オペランドに対する参照のことです。プログラム空間内の命令の説明は「第3章　命令セット」を参照してください。データ参照は、データ空間、プログラム・データをもつメモリ・セクションに対する参照です。命令ストリームのデータ・アイテムは、プログラム・カウンタ相対アドレッシング・モードでアクセスすることができ、これらのアクセスはプログラム参照として分類されています。3番目のタイプの外部参照は、コプロセッサ通信、割込みアクノリッジ・サイクル、およびブレークポイント・アクノリッジ・サイクルに使用され、CPU空間参照として分類されます。MC68030はプログラム空間、データ空間、または特殊機能の必要に応じて、CPU空間へアクセスするためのファンクション・コードを自動的に設定します。メモリ管理ユニットは、このファンクション・コードを使用して独立したプログラム(リード・オンリ)およびデータ(リード-ライト)メモリ領域を構成することができます。

本章ではMC68030のデータ構成およびアドレッシング機能について説明します。命令で使用するオペランドの種類をリストし、レジスタとそのオペランドとしての使い方を解説します。次にメモリ内のデータ構成とメモリ内のデータへアクセスするのに使用できるアドレッシング・モードを説明します。最後に、システム・スタックとユーザ・プログラム・スタックおよびキューについて述べます。

2. 1　命令オペランド

MC68030は多様なアプリケーションの要求に応えるために、汎用のオペランド・セットをサポートしています。MC68030の命令オペランドは、レジスタ、メモリ、または命令そのものに置くことができます。命令オペランドはコプロセッサに置くこともできます。オペランドのサイズは、1ビット、1から32ビット長のビット・フィールド、バイト(8ビット)、ワード(16ビット)、ロング・ワード(32ビット)、またはクワッド・ワード(64ビット)と定義されています。各命令のオペランドのサイズは、命令の中に明示されるか、命令の操作によって暗黙に決められています。コプロセッサは、データ・オペランドのタイプとサイズがきわめて特殊ながら大きく変化する必要のある計算モデルをサポートするように設計されています。そのため、コプロセッサ命令は任意のサイズのオペランドを指定できるようになっています。

2.2 レジスタ内のデータ構成

8個のデータ・レジスタは1、8、16、32、および64ビットのデータ・オペランド、16または32ビットのアドレス、および1 ~32ビットのビット・フィールドを扱うことができます。7個のアドレス・レジスタと3個のスタック・ポインタは、16または32ビットのアドレス・オペランドに使用可能です。制御レジスタ(SR、VBR、SFC、DFC、CACR、CAAR、CRP、SRP、TC、TT0、TT1およびMMUSR)は機能によって扱うサイズが異なります。コプロセッサは独自のオペランド・サイズを定義し、オンチップ・レジスタでそれらを扱うことができます。

ビット (0≦モジュロ(オフセット)<31、オフセット0=MSB)

31	30		2	1	0
MSB		・・・			LSB

バイト

31	23	15	7 0
上位バイト	中上位バイト	中下位バイト	下位バイト

16ビット・ワード

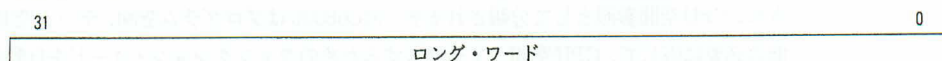
31	15 0
上位ワード	下位ワード

ロング・ワード

31 0
ロング・ワード

クワッド・ワード

63	32
MSB	任意のDy

31	0
任意のDy	LSB

ビット・フィールド(0≦オフセット<32、0<幅≦32)

31	幅			0
オフセット	MSB	・・・	LSB	

注:幅+オフセット<32の場合、ビット・フィールドはレジスタ内で循環します。

アンパック型式のBCD(a=MSB)

31	7	6	5	4	3	2	1	0
	x	x	x	x	a	b	c	d

パック型式のBCD(a=MSB第1デジット、e=MSB第2デジット)

31	7	6	5	4	3	2	1	0
	a	b	c	d	e	f	g	h

データ・レジスタ内のデータ構成

2.2.1 データ・レジスタ

各データ・レジスタの幅は32ビットです。バイト・オペランドは下位8ビット、ワード・オペランドは下位16ビット、そしてロング・ワード・オペランドは32ビット全体を使用します。データ・レジスタをソースまたはデスティネーションのオペランドとして使うときには、該当の下位バイトまたはワード(それぞれ、バイトまたはワードの操作に対応)だけが使用または変更されます。残りの上位部分は使用することも変更することもできません。ロング・ワード整数の最下位ビットは、ビット0としてアドレス指定され、最上位ビットはビット31としてアドレス指定されます。ビット・フィールドの場合、最上位ビットはビット0としてアドレス指定され、最下位ビットはフィールド幅−1としてアドレス指定されます。フィールド+オフセットの幅が32を超えると、ビット・フィールドはレジスタ内で循環します。次にデータ・レジスタ内の各種データ・タイプの構成を示します。

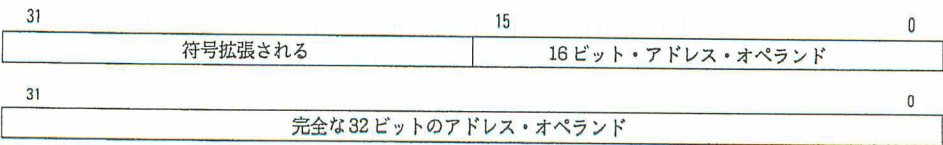
クワッド・ワードのデータは2つのロング・ワードで構成され、32ビットの乗算の積または32ビットの除算の商(符号付きおよび符号なし)に使われます。クワッド・ワードは任意の2個のデータ・レジスタで構成することができ、使用するレジスタの順序や組合せに関する制約はありません。このデータ・タイプを明示して使用する命令はありませんが、MOVEM命令を使用してレジスタ間でクワッド・ワードの転送を行なうことができます。

2進化10進(BCD)データは、2進形式で10進数を表現します。さまざまなBCDコードが考案されていますが、M68000ファミリのBCD命令は最下位4ビットが対応する10進数値をもつ2進数フォーマットをサポートします。2種類のBCDフォーマットを使用しています。アンパックBCDフォーマットでは1バイトが1桁で、最下位4ビットには2進数が入り最上位4ビットは未定義です。パックBCDフォーマットでは1バイトが2桁になっており、最下位4ビットには最下位桁が入ります。

2.2.2 アドレス・レジスタ

アドレス・レジスタおよびスタック・ポインタは、それぞれ32ビット幅で、32ビットのアドレスを保持します。アドレス・レジスタは、バイト・サイズのオペランドに使用することはできません。したがって、アドレス・レジスタをソース・オペランドとして使用する場合は、操作サイズに応じて下位ワードまたはロング・ワード・オペランド全体が使用されます。アドレス・レジスタをデスティネーション・オペランドとして使用する場合は、操作サイズには関係なくそのレジスタ全体が影響を受けます。ソース・オペランドがワード・サイズの場合、オペランドは32ビットに符号拡張されてから、アドレス・レジスタ・デスティネーションに対する操作に使用されます。アドレス・レジスタは、主にアドレスとアドレス計算のサポートに使用されます。命令セットには、アドレス・レジスタの内容を加算、減算、比較、および転送する命令が含まれています。次にアドレス・レジスタ内のアドレス構成を示します。

31	15	0
符号拡張される	16ビット・アドレス・オペランド	

31	0
完全な32ビットのアドレス・オペランド	

アドレス・レジスタ内のアドレス構成

2. 2. 3　制御レジスタ

ここで説明する制御レジスタは、スーパバイザ機能のための制御情報を保持し、サイズは可変です。ステータス・レジスタのユーザ部分(コンディション・コード・レジスタCCR)を除いては、スーパバイザ特権レベルの命令でしかアクセスされません。

図1-4に示すステータス・レジスタ(SR)は16ビット幅です。ステータス・レジスタの12ビットしか定義されていません。未定義の値はすべてモトローラが将来使用するため予約しています。未定義のビットは0で読み出されますが、書込みを行なうときは将来的に互換性が維持されるようにゼロを書き込まなければなりません。ステータス・レジスタの下位バイトは、コンディション・コード・レジスタ(CCR)になっています。コンディション・コード・レジスタに対する操作は、スーパバイザまたはユーザ特権レベルで実行することができます。ステータス・レジスタおよびコンディション・コード・レジスタに対する操作はワード・サイズ操作ですが、コンディション・コード・レジスタに対する操作では、特権レベルに関係なく上位バイトはすべて0で読み出され、書込みは無視されます。

スーパバイザ・プログラミング・モデル(図1-3)に制御レジスタを示します。キャッシュ制御レジスタ(CACR)は、オンチップ命令キャッシュおよびデータ・キャッシュに対して制御およびステータス情報を与えます。キャッシュ・アドレス・レジスタ(CAAR)は、キャッシュ制御機能に対するアドレスを保持します。ベクタ・ベース・レジスタ(VBR)は、例外ベクタ・テーブルのベース・アドレスを保持します。CACR、CAAR、およびVBR に関係するすべての操作は、これらのレジスタがソース・オペランドまたはデスティネーション・オペランドのいずれに使用する場合も、ロング・ワード操作となります。

オルタネート・ファンクション・コード・レジスタ(SFCおよびDFC)はビット2：0だけがインプリメントされた32ビット・レジスタで、これらのビットはMOVES、PLOAD、PFLUSH、およびPTEST命令のオペランドの読出しまたは書込みのときに、アドレス空間の値(FC0 ～FC2)を保持します。オルタネート・ファンクション・コード・レジスタとの間の転送はMOVEC命令で行なわれます。これらはロング・ワード転送ですが、上位29ビットは0で読み出され、書込みは無視されます。

スーパバイザ・プログラミング・モデルの残りの制御レジスタは、メモリ管理ユニットが使用します。CPUルート・ポインタ(CRP)およびスーパバイザ・ルート・ポインタ(CRP)は、ユーザおよびスーパバイザ・アドレス変換ツリーのポインタを保持します。これらの64ビット・レジスタとの間のデータ転送はクワッド・ワード転送です。変換制御レジスタ(TC)は、メモリ管理ユニットに対する制御情報を保持します。MC68030はこの32ビット・レジスタには常にロング・ワード転送を使用してアクセスします。トランスペアレント変換レジスタTT0およびTT1もそれぞれ32ビットのデータを保持しており、アドレス変換を行なわないダイレクト・アドレッシングのメモリ領域を区別しています。これらのレジスタ間のデータ転送はロング・ワード転送です。MMUステータス・レジスタ(MMUSR)は、PTEST命令実行後にMMUのステータスを格納します。これは16ビット・レジスタで、MMUSRとの間の転送はワード転送です。詳細については「第9章　メモリ管理ユニット」を参照してください。

2. 3　メモリ内のデータ構成

メモリはバイト・アドレス・ベースで構成されており、下位アドレスが上位バイトに対応しています。ロング・ワード・データ・アイテムのアドレスNは、その上位ワードの最上位バイトのアドレスに対応します。下位ワードのアドレスはN＋2 にあり、その最下位バイトのアドレスはN＋3

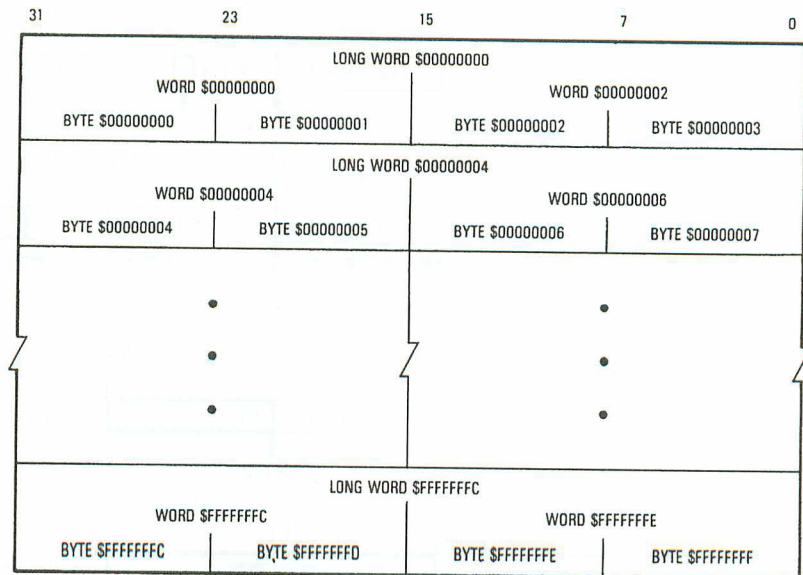



図2-1　メモリ・オペランドのアドレス

です(図2-1 参照)。

MC68030ではデータを偶数バイト境界に整列させる必要はありません(図2-2 参照)が、データがそのオペランド・サイズと同じバイト境界に整列していれば、データの転送効率が最大になります。しかし、命令ワードは偶数バイト境界に整列していなければなりません。

MC68030 はメモリ内で次のデータ・タイプをサポートしています。

●ビットおよびビット・フィールド・データ

●8、16、または32ビットの整数データ

●32ビットのアドレス

●2進化10進数(パック型式およびアンパック型式)

これらのデータ・タイプは、メモリ内では図2-2 のように構成されています。これらはすべて任意のバイト・アドレスでアクセスできます。

コプロセッサは、任意のデータ・タイプおよび255バイトまでのデータ長を扱うことができます。たとえば、浮動小数点演算コプロセッサMC68881/MC68882は、クワッド・ワード・サイズのデータ(倍精度浮動小数点数値)へのメモリ・アクセスをサポートしています。

ビット・オペランドは、メモリ内の1バイト(ベース・バイト)を選択するベース・アドレスと、そのバイトの中の1ビットを選択するビット番号によって指定されます。このバイトの最上位ビットのビット番号は7です。

ビット・フィールドのデータは、次のもので指定されます。

1. メモリ内の1バイトを選択するベース・アドレス
2. ベース・バイトの最上位ビットに関連してビット・フィールドの最左端(ベース)ビットを示すビット・フィールド・オフセット
3. そのビット・フィールド内のベース・ビットから右側のビット数を示すビット・フィールド幅

ベース・バイトの最上位ビットはビット・フィールド・オフセット0、ベース・バイトの最下位ビットはビット・フィールド・オフセット7、そしてメモリ内の直前のバイトの最下位ビットはビット・オフセット−1です。ビット・フィールドは-2^{31}～$2^{31}-1$の範囲の値をとることができ、ビット・フィールド幅は1 ～32ビットの範囲の値をとることができます。

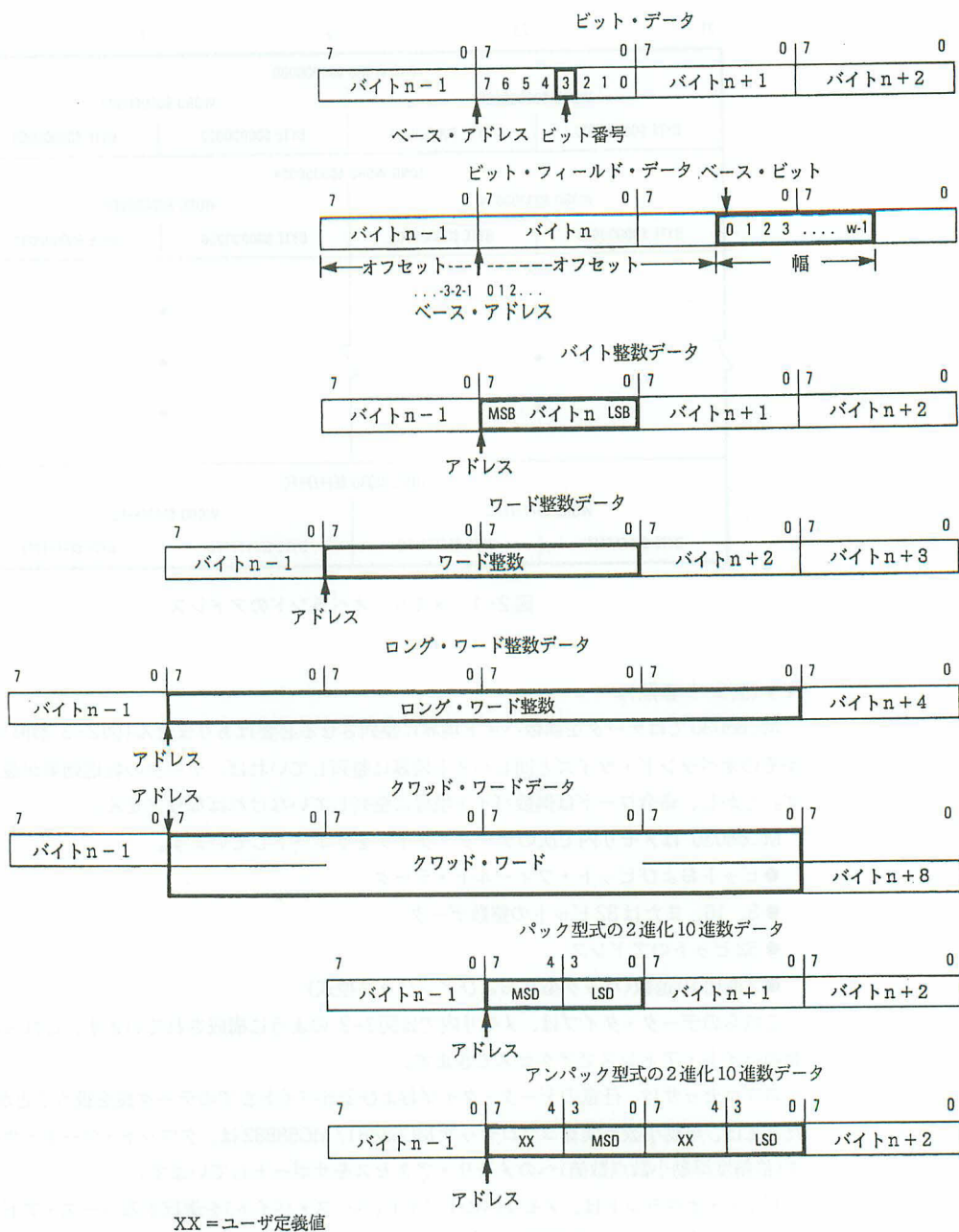



図2-2　メモリ内のデータ構成

2. 4　　アドレッシング・モード

命令のアドレッシング・モードでオペランドの値を指定したり(イミディエイト・オペランド)、オペランドを含むレジスタを指定したり(レジスタ直接アドレッシング・モード)、あるいはメモリ内のオペランドの実効アドレスを生成する方法を指定することができます。各アドレッシング・モードに対してアセンブラのシンタックスが定義されています。

図2-3に単一実効アドレス命令のオペレーション・ワードの一般フォーマットを示します。実効アドレス・フィールドは、多数の定義モードの1つを使用できるオペランドのアドレッシング・モー

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X	X	実効アドレス					
										モード			レジスタ		

図2-3　単一実効アドレス命令のオペレーション・ワード

ドを指定します。実効アドレスは2つの3ビット・フィールド、すなわち、モード・フィールドとレジスタ・フィールドで構成されています。モード・フィールドの値は、アドレッシング・モードの1つまたは1つの組合せを選択します。レジスタ・フィールドはそのモードのレジスタまたはレジスタを使用しないモードに対するサブモードを指定します。

多くの命令がオペランドの1つに対するアドレッシング・モードを暗黙に指定します。これらの命令のフォーマットには、1つのアドレッシング・モードしか使用しないオペランドに対する適切なフィールドが含まれています。

実効アドレス・フィールドで、オペランドのアドレスを完全に指定するために、追加の情報が必要になることもあります。この追加情報は実効アドレス拡張とよばれ、オペレーション・ワードの次の1ワードまたは複数のワードに含まれていて、命令の一部とみなされます。拡張ワード・フォーマットの詳細については、「2. 5　実効アドレス・エンコーディングの概要」を参照してください。

本章ではアドレッシング・モードを説明するのに、次のような表記法を用いています。

EA ——実効アドレス

An ——アドレス・レジスタn

例：A3はアドレス・レジスタ3

Dn ——データ・レジスタn

例：D5はデータ・レジスタ5

Xn. SIZE * SCALE ——インデックス・レジスタn（データまたはアドレス）、インデックス・サイズ（ワードはW、ロング・ワードはL）、およびスケール・ファクタ（1、2、4、8はそれぞれ、なし、ワード、ロング・ワード、クワッド・ワードのスケーリングに対応）を示します。

PC ——プログラム・カウンタ

d_n ——ディスプレースメント値、nビット幅

bd ——ベース・ディスプレースメント

od ——アウタ・ディスプレースメント

L ——ロング・ワード・サイズ

W ——ワード・サイズ

( ) ——レジスタ内の間接アドレス

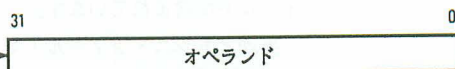
[ ] ——メモリ内の間接アドレス

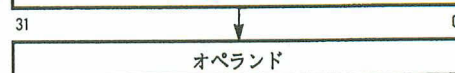
アドレッシング・モードでレジスタを使用するときは、操作ワードのレジスタ・フィールドで使用するレジスタを指定します。命令の中の他のフィールドは、選択されたレジスタがアドレス・レジスタかデータ・レジスタか、そしてそのレジスタをどのように使用するかを指定します。

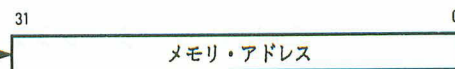
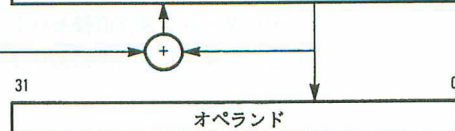
2. 4. 1　データ・レジスタ直接モード

データ・レジスタ直接モードでは、オペランドは実効アドレス・レジスタ・フィールドで指定されるデータ・レジスタにあります。

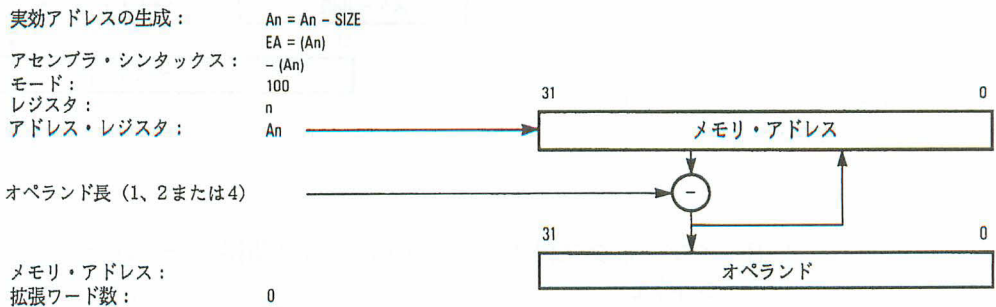
2. 4. 2　アドレス・レジスタ直接モード

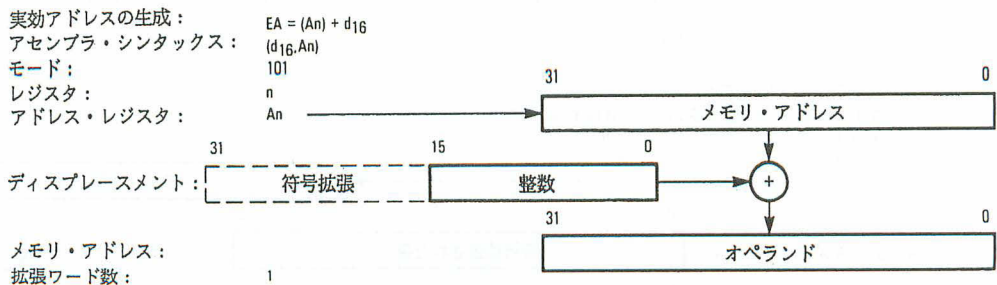
アドレス・レジスタ直接モードでは、オペランドは実効アドレス・レジスタ・フィールドで指定されるアドレス・レジスタにあります。

2. 4. 3　アドレス・レジスタ間接モード

アドレス・レジスタ間接モードでは、オペランドはメモリにあり、そのオペランドのアドレスはレジスタ・フィールドで指定されるアドレス・レジスタにあります。

2. 4. 4　ポストインクリメント付きアドレス・レジスタ間接モード

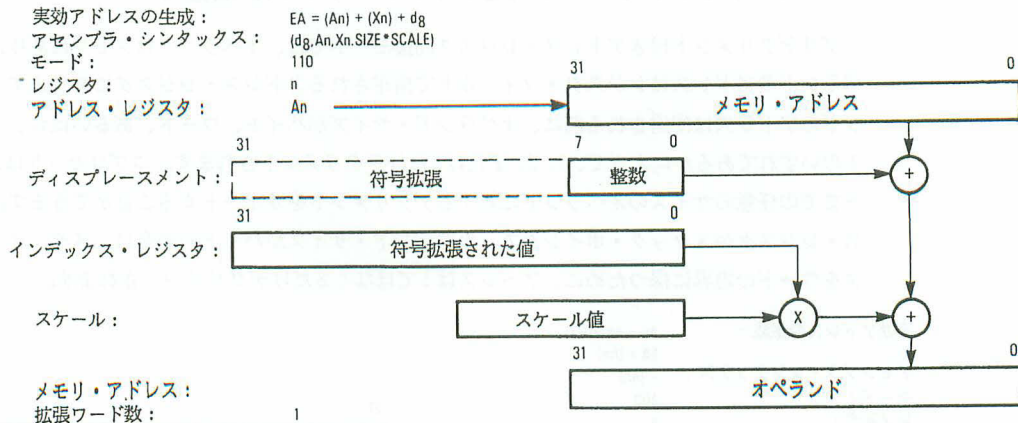
ポストインクリメント付きアドレス・レジスタ間接モードでは、オペランドはメモリにあり、そのオペランドのアドレスはレジスタ・フィールドで指定されるアドレス・レジスタにあります。オペランドのアドレスは、使用された後そのオペランドのサイズがバイト、ワード、あるいはロング・

ワードのいずれであるかによって、1、2、または4だけインクリメントされます。コプロセッサは255バイトまでの任意のサイズのオペランドに対するインクリメントをサポートすることができます。アドレス・レジスタがスタック・ポインタで、オペランド・サイズがバイトの場合は、スタック・ポインタをワードの境界に保つために、アドレスは1ではなく2だけインクリメントされます。

2.4.5　プリデクリメント付きアドレス・レジスタ間接モード

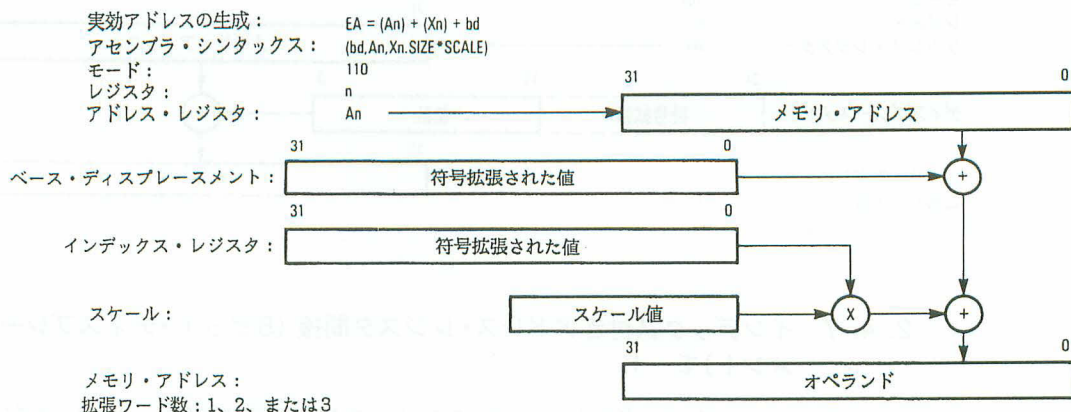
プリデクリメント付きアドレス・レジスタ間接モードでは、オペランドはメモリにあり、そのオペランドのアドレスはレジスタ・フィールドで指定されるアドレス・レジスタにあります。オペランドのアドレスは使用される前に、オペランド・サイズがバイト、ワード、あるいはロング・ワードのいずれであるかによって、1、2、または4だけデクリメントされます。コプロセッサは255バイトまでの任意のサイズのオペランドに対するデクリメントをサポートすることができます。アドレス・レジスタがスタック・ポインタで、オペランド・サイズがバイトの場合は、スタック・ポインタをワードの境界に保つために、アドレスは1ではなく2だけデクリメントされます。

2.4.6　ディスプレースメント付きアドレス・レジスタ間接モード

ディスプレースメント付きアドレス・レジスタ間接モードでは、オペランドはメモリにあります。オペランドのアドレスは、アドレス・レジスタ内のアドレスと拡張ワード内の符号拡張された16ビット・ディスプレースメント整数の和になります。ディスプレースメントは、常に実効アドレスの計算に使用される前に32ビットに符号拡張されます。

2.4.7　インデックス付きアドレス・レジスタ間接（8ビット・ディスプレースメント）モード

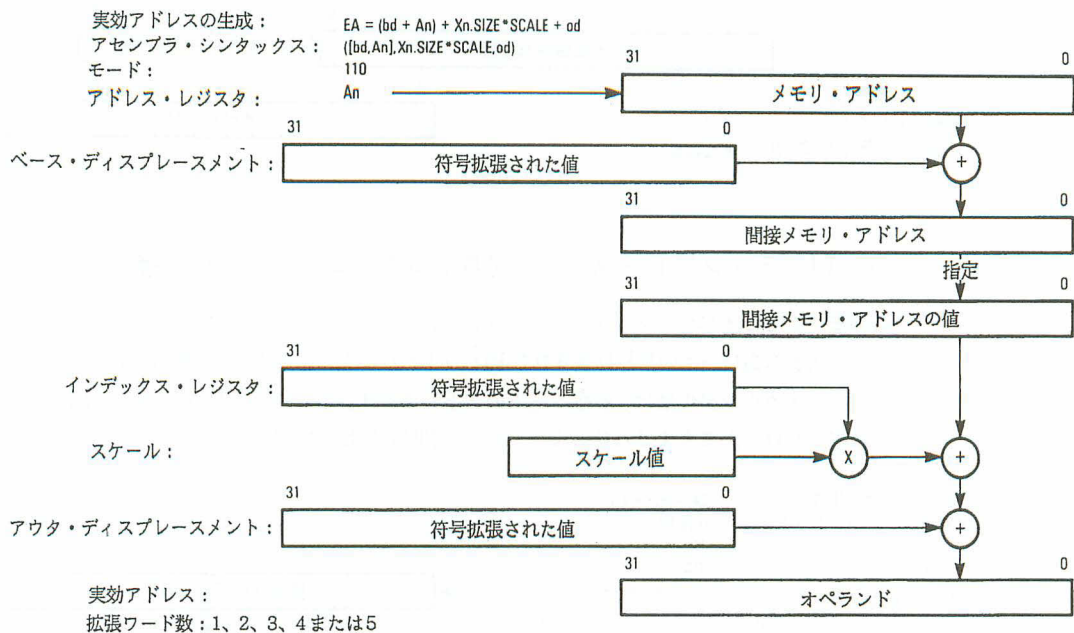
このアドレッシング・モードでは、インデックス・レジスタ指示子および8ビット・ディスプレースメントを含む1つの拡張ワードが必要です。インデックス・レジスタ指示子にはサイズおよびス

ケール情報が含まれます。このモードではオペランドはメモリにあります。オペランドのアドレスは、アドレス・レジスタ、拡張ワードの下位8ビットの符号拡張されたディスプレースメント値、およびインデックス・レジスタの符号拡張された内容(スケールされている場合もある)の和になります。このモードでは、ユーザがディスプレースメント、アドレス・レジスタ、そしてインデックス・レジスタを指定しなければなりません。

2. 4. 8 インデックス付きアドレス・レジスタ間接(ベース・ディスプレースメント)モード

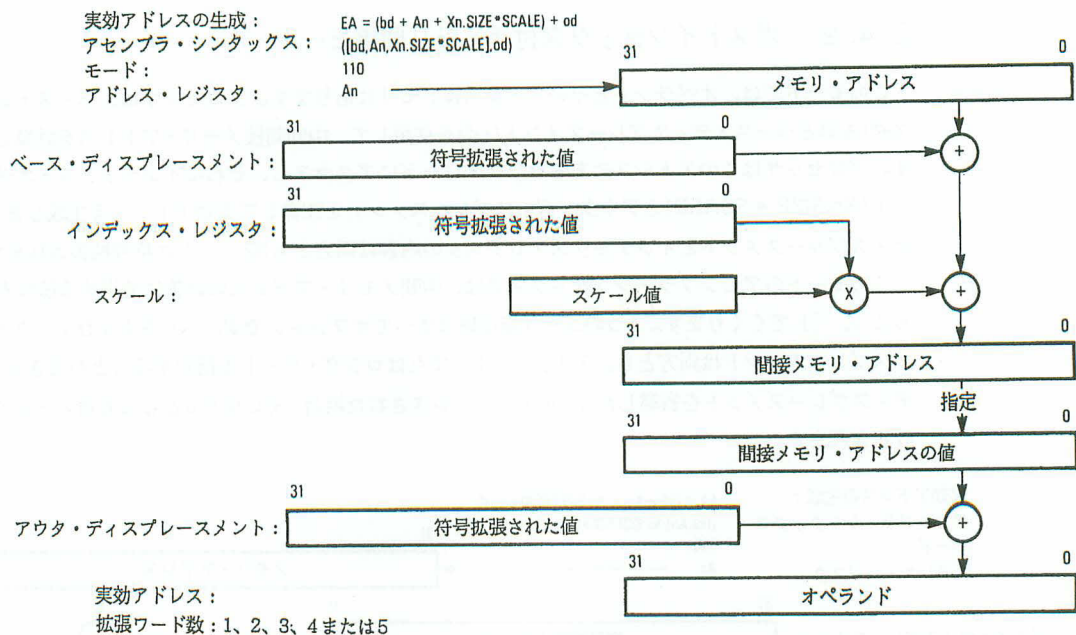
このアドレッシング・モードでは、インデックス・レジスタ指示子とオプションの16または32ビットの符号拡張されたベース・ディスプレースメントが必要です。インデックス・レジスタ指示子にはサイズおよびスケール情報が含まれます。オペランドはメモリにあります。オペランドのアドレスは、アドレス・レジスタの内容、符号拡張されたインデックス・レジスタの内容をスケールした値、およびベース・ディスプレースメントの和になります。

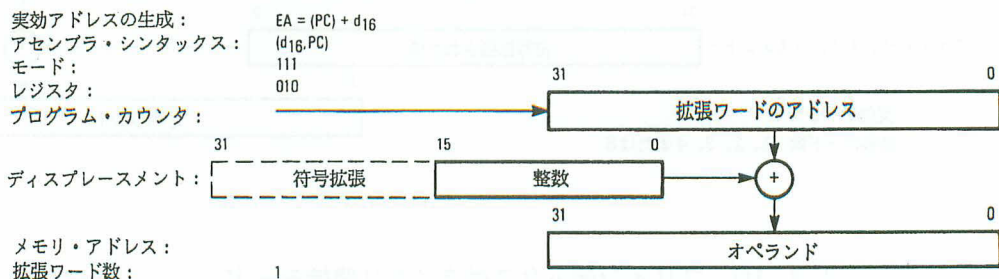
このモードでは、アドレス・レジスタ、インデックス・レジスタ、およびディスプレースメントの指示はすべてオプションです。どれも指定されなかった場合、実効アドレスは0になります。アドレス・レジスタが指定されてなく、インデックス・レジスタがデータ・レジスタ(Dn)の場合、データ・レジスタ間接アドレスとなります。

2. 4. 9　ポストインデックス付きメモリ間接モード

このモードでは、オペランドとそのアドレスはメモリにあります。プロセッサは、ベース・レジスタ(An)とベース・ディスプレースメント(bd)を使用して、中間間接メモリ・アドレスを計算します。プロセッサはこのアドレスにあるロング・ワードにアクセスし、それにインデックス・オペランド(Xn.SIZE ＊ SCALE)とアウタ・ディスプレースメントを加算して実効アドレスを生成します。ディスプレースメントとインデックス・レジスタの内容は両方とも32ビットに符号拡張されます。

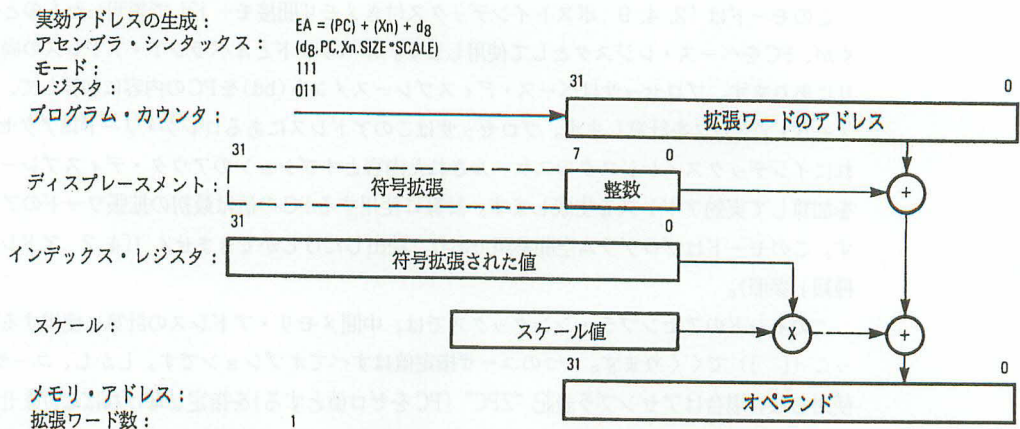
このモードのアセンブラ・シンタックスでは、中間メモリ・アドレスの計算に使用する値は大かっこ（[　]）でくくります。4つのユーザ指定値はすべてオプションです。ベースおよびアウタ・ディスプレースメントは両方とも、ヌル、ワード、またはロング・ワードを指定することができます。ディスプレースメントを省略したり、要素がサプレスされた場合、その値を0として実効アドレスが計算されます。

2. 4. 10　プリインデックス付きメモリ間接モード

このモードでは、オペランドとそのアドレスはメモリにあります。プロセッサはベース・レジスタ(An)、ベース・ディスプレースメント(bd)、およびインデックス・オペランド(Xn.SIZE ＊ SCALE)を使用して、中間間接メモリ・アドレスを計算します。プロセッサはこのアドレスにあるロング・ワードにアクセスし、それにアウタ・ディスプレースメントを加算して実効アドレスを生成します。ディスプレースメントとインデックス・レジスタの内容は両方とも32ビットに符号拡張されます。

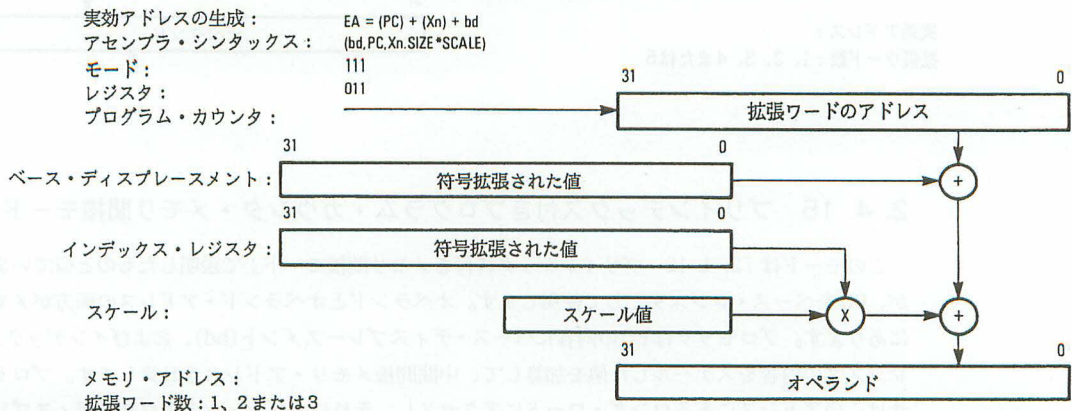
このモードのアセンブラ・シンタックスでは、中間メモリ・アドレスの計算に使用する値は大かっこ（[　]）でくくります。4つのユーザ指定値はすべてオプションです。ベースおよびアウタ・ディスプレースメントは両方とも、ヌル、ワード、またはロング・ワードを指定することができます。ディスプレースメントを省略したり、要素がサプレスされた場合、その値を0として実効アドレスが計算されます。

2. 4. 11 ディスプレースメント付きプログラム・カウンタ間接モード

このモードではオペランドはメモリにあります。オペランドのアドレスはプログラム・カウンタのアドレスと拡張ワードの符号拡張された16ビット・ディスプレースメント整数との和になります。プログラム・カウンタの値は、拡張ワードのアドレスです。このモードはプログラム空間参照であり、読出しだけしかできません(「4. 2 アドレス空間の種類」参照)。

2. 4. 12 インデックス付きプログラム・カウンタ間接(8ビット・ディスプレースメント)モード

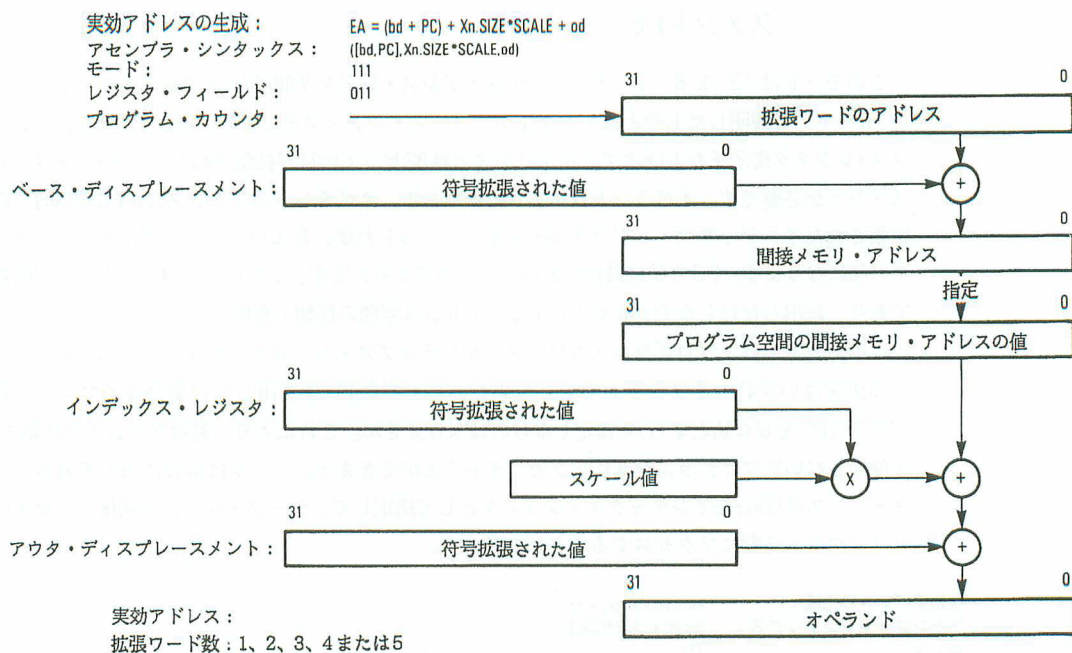
このモードは、「2. 4. 7 インデックス付きアドレス・レジスタ間接(8ビット・ディスプレースメント)モード」で説明したモードと似ていますが、ベース・レジスタとしてPCを使用します。オペランドはメモリにあります。オペランドのアドレスは、プログラム・カウンタのアドレス、拡張ワードの下位8ビットの符号拡張されたディスプレースメント整数、およびサイズおよびスケール付きの符号拡張されたインデックス・オペランドの和になります。PCの値は拡張ワードのアドレスです。このモードはプログラム空間参照であり、読出しだけしかできません。このアドレッシング・モー

ドを指定する場合、ユーザはディスプレースメント、PC、およびインデックス・レジスタを指定しなければなりません。

2. 4. 13　インデックス付きプログラム・カウンタ間接(ベース・ディスプレースメント)モード

このモードは「2. 4. 8　インデックス付きアドレス・レジスタ間接(ベース・ディスプレースメント)モード」で説明したものと似ていますが、ベース・レジスタとしてPCを使用します。インデックス・レジスタ指示子およびオプションの16または32ビットの符号拡張されたベース・ディスプレースメントが必要です。オペランドはメモリにあります。オペランドのアドレスは、PCの内容、符号拡張されたインデックス・レジスタの内容をスケールした値、およびベース・ディスプレースメントの和になります。PCの値は最初の拡張ワードのアドレスです。このモードはプログラム空間参照であり、読出しだけしかできません(「4. 2　アドレス空間の種類」参照)。

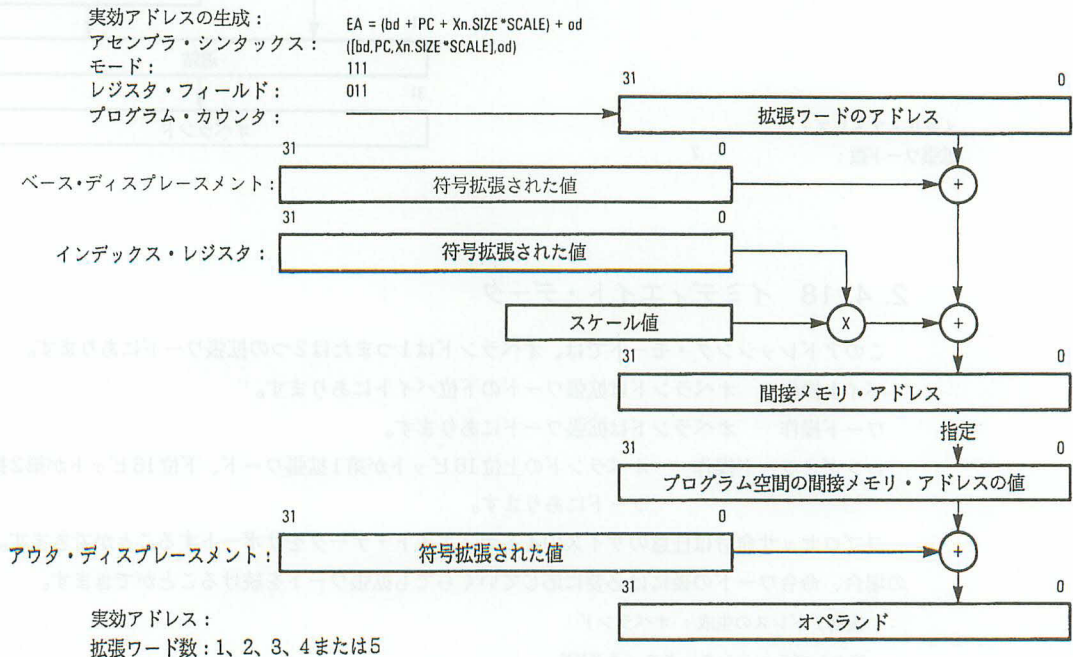
このモードでは、プログラム・カウンタ、インデックス・レジスタ、およびディスプレースメントの指定はいずれもオプションです。しかし、ユーザがPCを使用しない場合はアセンブラ表記“ZPC”(PCをゼロ値とする)を指定しなければなりません。これにより、実効アドレスの計算にPCを使用しないでプログラム空間にアクセスすることができます。ユーザは命令にZPCを置き、データ・レジスタ(Dn)をインデックス・レジスタとして指定して、データ・レジスタ間接アクセスによりプログラム空間にアクセスすることができます。

2. 4. 14　ポストインデックス付きプログラム・カウンタ・メモリ間接モード

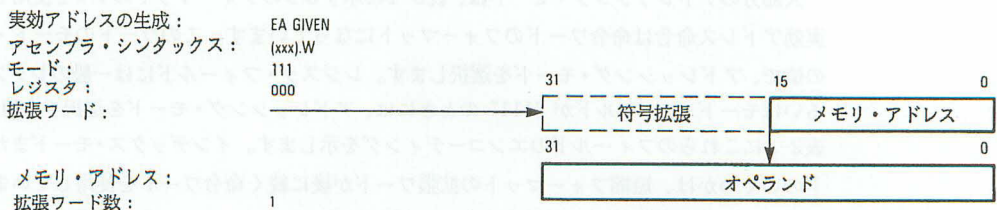
このモードは「2. 4. 9　ポストインデックス付きメモリ間接モード」で説明したものと似ていますが、PCをベース・レジスタとして使用します。オペランドとオペランド・アドレスの両方がメモリにあります。プロセッサはベース・ディスプレースメント(bd)をPCの内容に加算して、中間間接メモリ・アドレスを計算します。プロセッサはこのアドレスにあるロング・ワードにアクセスし、それにインデックス・レジスタのスケールされた内容とオプションのアウタ・ディスプレースメントを加算して実効アドレスを生成します。計算に使用するPCの値は最初の拡張ワードのアドレスです。このモードはプログラム空間参照であり、読出しだけしかできません（「4. 2　アドレス空間の種類」参照)。

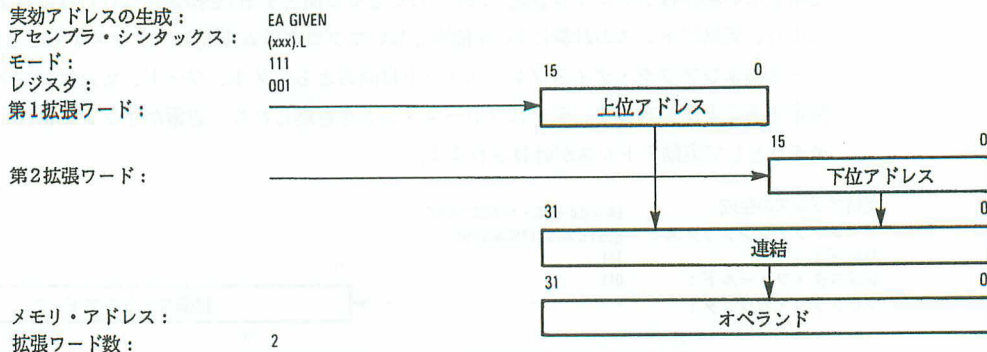
このモードのアセンブラ・シンタックスでは、中間メモリ・アドレスの計算に使用する値は大かっこ（[　]）でくくります。4つのユーザ指定値はすべてオプションです。しかし、ユーザがPCを使用しない場合はアセンブラ表記“ZPC”（PCをゼロ値とする)を指定しなければなりません。これにより、実効アドレスの計算にPCを使用しないでプログラム空間にアクセスすることができます。ベースおよびアウタ・ディスプレースメントは両方とも、ヌル、ワード、またはロング・ワードを指定することができます。ディスプレースメントを省略したり、要素がサプレスされた場合、その値を0として実効アドレスが計算されます。

2. 4. 15　プリインデックス付きプログラム・カウンタ・メモリ間接モード

このモードは「2. 4. 10　プリインデックス付きメモリ間接モード」で説明したものと似ていますが、PCをベース・レジスタとして使用します。オペランドとオペランド・アドレスの両方がメモリにあります。プロセッサはPCの内容にベース・ディスプレースメント(bd)、およびインデックス・レジスタの内容をスケールした値を加算して、中間間接メモリ・アドレスを計算します。プロセッサはこのアドレスにあるロング・ワードにアクセスし、それにオプションのアウタ・ディスプレー

スメントを加算して実効アドレスを生成します。PCの値は最初の拡張ワードのアドレスです。このモードはプログラム空間参照であり、読出しだけしかできません（「4. 2　アドレス空間の種類」参照）。

このモードのアセンブラ・シンタックスでは、中間メモリ・アドレスの計算に使用する値は大かっこ（[　]）でくくります。4つのユーザ指定値はすべてオプションです。しかし、ユーザがPCを使用しない場合はアセンブラ表記“ZPC”（PCをゼロ値とする)を指定しなければなりません。これにより、実効アドレスの計算にPCを使用しないでプログラム空間にアクセスすることができます。ベースおよびアウタ・ディスプレースメントは両方とも、ヌル、ワード、またはロング・ワードを指定することができます。ディスプレースメントを省略したり、要素がサプレスされた場合、その値を0として実効アドレスが計算されます。

2. 4. 16　絶対ショート・アドレス・モード

このアドレス・モードでは、オペランドはメモリにあり、オペランドのアドレスは拡張ワードにあります。16ビットのアドレスは使用する前に32ビットに符号拡張されます。

2.4.17　絶対ロング・アドレス・モード

このモードではオペランドはメモリにあり、オペランドのアドレスは、メモリ内で命令ワードの後の2つの拡張ワードを占有します。最初の拡張ワードにはアドレスの上位部分、そして2番目の拡張ワードには下位部分が入ります。

2.4.18　イミディエイト・データ

このアドレッシング・モードでは、オペランドは1つまたは2つの拡張ワードにあります。

バイト操作　　オペランドは拡張ワードの下位バイトにあります。

ワード操作　　オペランドは拡張ワードにあります。

ロング・ワード操作　　オペランドの上位16ビットが第1拡張ワード、下位16ビットが第2拡張ワードにあります。

コプロセッサ命令は任意のサイズのイミディエイト・データをサポートすることができます。この場合、命令ワードの後には必要に応じていくらでも拡張ワードを続けることができます。

実効アドレスの生成：オペランド

アセンブラ・シンタックス：＃XXX

モード・フィールド：111

レジスタ・フィールド：100

拡張ワード数：コプロセッサ命令を除いて1または2

2.5　実効アドレス・エンコーディングの概要

大部分のアドレッシング・モードは、表2-1に示す3つのフォーマットの1つを使用します。単一実効アドレス命令は命令ワードのフォーマットになっています。このワードのモード・フィールドの値で、アドレッシング・モードを選択します。レジスタ・フィールドには一般のレジスタ番号、あるいはモード・フィールドが“111”のときには、アドレッシング・モードを選択する値があります。表2-3にこれらのフィールドのエンコーディングを示します。インデックス・モードまたは間接モードのいくつかは、短縮フォーマットの拡張ワードが後に続く命令ワードを使用しています。他のインデックスまたは間接モードは、命令ワードおよびフル・フォーマットの拡張ワードで構成されます。MC68030 の最長命令は10個の拡張ワードをもちます。これは、ソースおよびデスティネーション両方の実効アドレスに対して、フル・フォーマットの拡張ワード、そして同じく両方のアドレスに対して32ビットのベース・ディスプレースメントと32ビットのアウタ・ディスプレースメント

表2-1　実効アドレスの指定フォーマット

単一実効アドレス命令フォーマット

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
×	×	×	×	×	×	×	×	×	×	実効アドレス					
										モード			レジスタ		

短縮フォーマットの拡張ワード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D/A	レジスタ			W/L	スケール		0	ディスプレースメント							

フル・フォーマットの拡張ワード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D/A	レジスタ			W/L	スケール		1	BS	IS	BDサイズ		0	I/IS		
ベース・ディスプレースメント(0、1または2ワード)															
アウタ・ディスプレースメント(0、1または2ワード)															

フィールド	定　義
命令：	
レジスタ	一般レジスタ番号
拡張：	
レジスタ	インデックス・レジスタ番号
D/A	インデックス・レジスタのタイプ 0＝Dn 1＝An
W/L	ワード/ロング・ワードのインデックス・サイズ 0＝符号拡張ワード 1＝ロング・ワード
スケール	スケール・ファクタ 00＝1 01＝2 10＝4 11＝8

フィールド	定　義
BS	ベース・レジスタのサプレス： 0＝ベース・レジスタを加算 1＝ベース・レジスタをサプレス
IS	インデックスのサプレス 0＝インデックス・オペランドを評価して加算 1＝インデックス・オペランドをサプレス
BD SIZE	ベース・ディスプレースメントのサイズ 00＝予約 01＝ヌル・ディスプレースメント 10＝ワード・ディスプレースメント 11＝ロング・ワード・ディスプレースメント
I/IS	インデックス/間接の選択： ビット6、インデックスのサプレスに関連して決定される間接またはインデックスのオペランド

を使用したMOVE命令です。ただし、コプロセッサ命令には拡張ワードがいくつあってもかまいません。「第10章　コプロセッサ・インタフェースの説明」を参照してください。

フル・フォーマットを使用する実効アドレスに対しては、インデックス・サプレス(IS)ビットおよびインデックス/間接の選択(I/IS)フィールドの組合せにより、インデックスと間接指定のタイプが決まります。表2-2に、ISおよびI/IS値に対応するインデックスおよび間接操作を一覧にして示します。

実効アドレス・モードは使用方法によってグループにまとめられます。これらは次のように分類することができます。

データ　　データ・アドレッシング実効アドレス・モードは、データ・オペランドを参照するモードです。

メモリ　　メモリ・アドレッシング実効アドレス・モードは、メモリ・オペランドを参照するモードです。

可　変　　可変アドレッシング実効アドレス・モードは、可変(書換え可能)オペランドを参照するモードです。

第2章　データ構成およびアドレッシング機能

表2-2 IS-I/ISメモリ間接指定エンコーディング

IS	インデックス/間接	操作
0	000	メモリ間接なし
0	001	ヌル・アウタ・ディスプレースメント、プリインデックス付き間接
0	010	ワード・アウタ・ディスプレースメント、プリインデックス付き間接
0	011	ロング・アウタ・ディスプレースメント、プリインデックス付き間接
0	100	予約
0	101	ヌル・アウタ・ディスプレースメント、ポストインデックス付き間接
0	110	ワード・アウタ・ディスプレースメント、ポストインデックス付き間接
0	111	ロング・アウタ・ディスプレースメント、ポストインデックス付き間接
1	000	メモリ間接なし
1	001	ヌル・アウタ・ディスプレースメント付きメモリ間接
1	010	ワード・アウタ・ディスプレースメント付きメモリ間接
1	011	ロング・アウタ・ディスプレースメント付きメモリ間接
1	100－111	予約

制　御　制御アドレッシング実効アドレス・モードは、サイズに関係しないメモリ・オペランドを参照するモードです。

表2-3に各実効アドレッシング・モードが属するカテゴリを示します。

これらのカテゴリを組み合わせることにより、さらにいくつかのより限定した新しいカテゴリを作ることができます。2つを組み合わせたものに、可変メモリまたはデータ可変があります。前者は可変とメモリの両方のアドレッシング・モードを意味し、後者はデータと可変の両方のアドレッシング・モードを意味します。

2.6　プログラマからみたアドレッシング・モード

インデックス・アドレッシング・モード、間接アドレッシング、および完全な32ビット・ディスプレースメントの拡張により、MC68020とMC68030の両方に対してプログラミング機能が一層強化されています。本節では、これらの機能をフルに引き出すアドレッシング・テクニックを紹介し、プログラマの観点からとらえたアドレッシング・モードを要約します。

ここで述べるアドレッシング・テクニックのいくつかは、データ・レジスタとアドレス・レジスタを交換可能な形で使用しています。MC68030はこれらの機能を実現すると同時に、アドレス・レジスタを使用したアドレッシングの性能を最適化しています。アドレスの計算にアドレス・レジスタを使用するプログラムの性能は、同様な形態でデータ・レジスタを使用するものより優れています。データ・レジスタによるアドレスの指定は、特に性能を最大限に高める必要のあるプログラムでは、できるだけ使わないようにしなければなりません。

2.6.1　アドレッシング機能

MC68020とMC68030の両方において、完全フォーマット拡張ワード(表2-1)のベース・レジスタ・サプレス(BS)ビットをセットすると、実効アドレスの計算にベース・アドレス・レジスタを含めないようにすることができます。これによって、ベース・レジスタの代わりに任意のインデックス・レジスタを使用できます。どのデータ・レジスタでもインデックス・レジスタとして使用でき

表2-3　実効アドレッシング・モードのカテゴリ

アドレス・モード	モード	レジスタ	データ	メモリ	制御	可変	アセンブラ・シンタックス
データ・レジスタ直接	000	reg. no.	×	–	–	×	Dn
アドレス・レジスタ直接	001	reg. no.	–	–	–	×	An
アドレス・レジスタ間接	010	reg. no.	×	×	×	×	(An)
ポストインクリメント付き アドレス・レジスタ間接	011	reg. no.	×	×	–	×	(An) +
プリデクリメント付き アドレス・レジスタ間接	100	reg. no.	×	×	–	×	– (An)
ディスプレースメント付き アドレス・レジスタ間接	101	reg. no.	×	×	×	×	(d_{16}, An)
インデックス(8ビット・ディスプレースメント)付きアドレス・レジスタ間接	110	reg. no.	×	×	×	×	(d_8, An, Xn)
インデックス(ベース・ディスプレースメント)付きアドレス・レジスタ間接	110	reg. no.	×	×	×	×	(bd, An, Xn)
ポストインデックス付きメモリ間接	110	reg. no.	×	×	×	×	([bd, An], Xn, od)
プリインデックス付きメモリ間接	110	reg. no.	×	×	×	×	([bd, An, Xn], od)
絶対ショート	111	000	×	×	×	×	(xxx).W
絶対ロング	111	001	×	×	×	×	(xxx).L
ディスプレースメント付き プログラム・カウンタ間接	111	010	×	×	×	–	(d_{16}, PC)
インデックス(8ビット・ディスプレースメント)付きプログラム・カウンタ間接	111	011	×	×	×	–	(d_8, PC, Xn)
インデックス(ベース・ディスプレースメント)付きプログラム・カウンタ間接	111	011	×	×	×	–	(bd, PC, Xn)
ポストインデックス付きPCメモリ間接	111	011	×	×	×	–	([bd, PC], Xn, od)
プリインデックス付きPCメモリ間接	111	011	×	×	×	–	([bd,PC, Xn], od)
イミディエイト	111	100	×	×	–	–	# 〈data

るため、データ・レジスタ間接形式(Dn)が可能です。このモードは任意のデータ・レジスタまたはアドレス・レジスタを使用できるという意味から、レジスタ間接(Rn)とよぶことができます。このアドレッシング・モードは、MC68030とMC68020がメモリのアドレス指定を行なうのに、データ・レジスタとアドレス・レジスタの両方を使用できるため、M68000ファミリの拡張となっています。これらのモードでインデックス・レジスタのサイズとスケールの指定(Xn.SIZE＊SCALE)を行なう能力により、アドレッシングの柔軟性が向上しています。SIZEパラメータを使用すれば、インデックス・レジスタの全体の内容を使用するか、最下位ワードを符号拡張して32ビットのインデックス値を求めることができます(図2-4参照)。

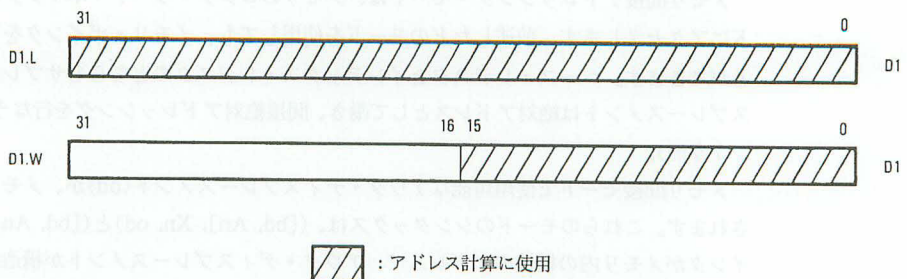



図2-4　SIZEによるインデックスの選択

シンタックス：(bd, An, Rn)

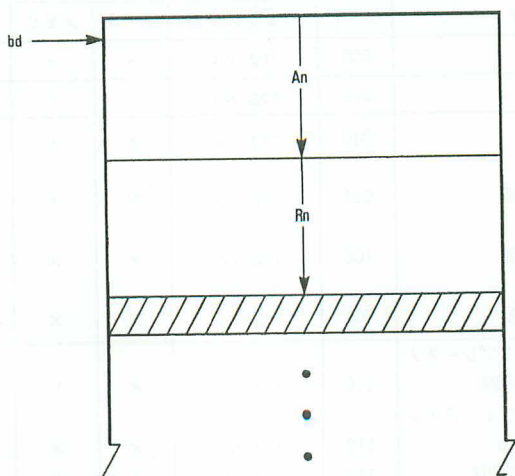



図2-5　インデックスによる絶対アドレスの使用

MC68020とMC68030の両方とも、さらにレジスタ間接モードを拡張することができます。ディスプレースメントは32ビット幅が使用できますので、絶対アドレスまたは絶対アドレスを含む式の結果を表現することができます。これにより、ベース・レジスタがサプレスされていないときは、(bd, Rn)または(bd, An, Rn)の一般レジスタ間接形式が得られます。したがって、絶対アドレスを1つまたは2つのレジスタで直接インデックスすることができます(図2-5参照)。

スケーリングは、実効アドレス計算に使用する前にインデックス・レジスタを左へ0、1、2、または3ビットだけ任意にシフトするオプションです(インデックス・レジスタ内の実際の値は変化しません)。これはレジスタを1、2、4、または8倍することと同じで、16個の汎用レジスタいずれかにある算術値を使用して、スケーリング値で決まるサイズの配列要素を直接添字付きで指定するためのものです。スケーリングを行なっても実効アドレスの計算時間は増えません。しかし、適当な派生モードと組み合わせて、さらに新しい機能をつくることができます。配列型の構造を絶対アドレスで指定し、ついで添字を付けることができます(例：bd, Rn * scale)。オプションとして、ダイナミック・ディスプレースメントをもつアドレス・レジスタをアドレス計算に含めることができます(例：bd, An, Rn * scale)。(An, Rn * scale)など、別のバリエーションも可能です。最初の例では、配列アドレスは図2-6に示すように、レジスタの内容とディスプレースメントの和になります。2番目の例では、Anは配列のアドレスをもちRnは添字をもっています。

メモリ間接アドレッシング・モードは、メモリのロング・ワード・ポインタを使用してオペランドにアクセスします。前述したどのモードを使用しても、メモリ・ポインタをアドレス指定することができます。ベース・レジスタとインデックス・レジスタのどちらもサプレスできるため、ディスプレースメントは絶対アドレスとして働き、間接絶対アドレッシングを行なうことができます(図2-7参照)。

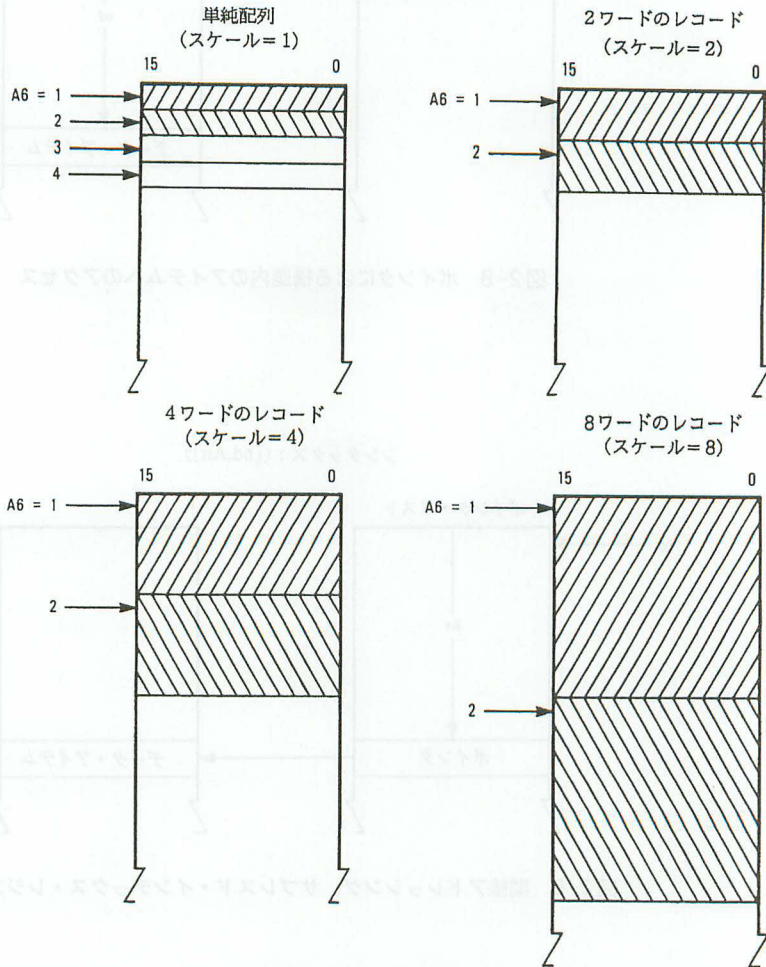
メモリ間接モードで使用可能なアウタ・ディスプレースメント(od)が、メモリのポインタに付加されます。これらのモードのシンタックスは、([bd, An], Xn, od)と([bd, An, Xn]、od)です。ポインタがメモリ内の構造のアドレスで、アウタ・ディスプレースメントが構造内のアイテムのオフセットのときには、メモリ間接モードで効率よくアイテムにアクセスすることができます（図2-8参照)。

注：配列の構造には関係なく、ソフトウェアは適当な値だけインデックスをインクリメントして、次のレコードを指します。

図2-6　配列アイテムのアドレッシング

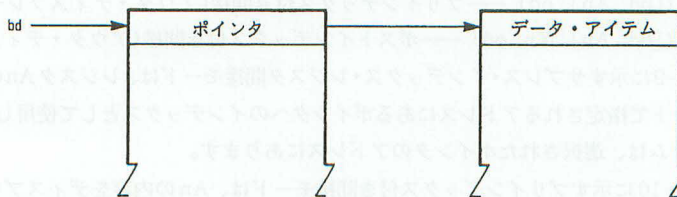



図2-7　間接絶対メモリ・アドレッシング

シンタックス：([An],od)

メモリ

構造

An

ポインタ

od

データ・アイテム

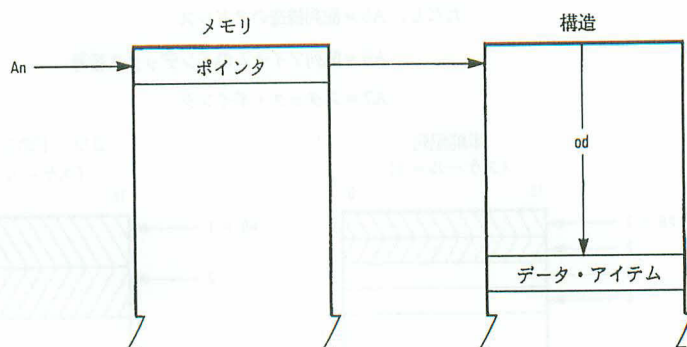



図2-8　ポインタによる構造内のアイテムへのアクセス

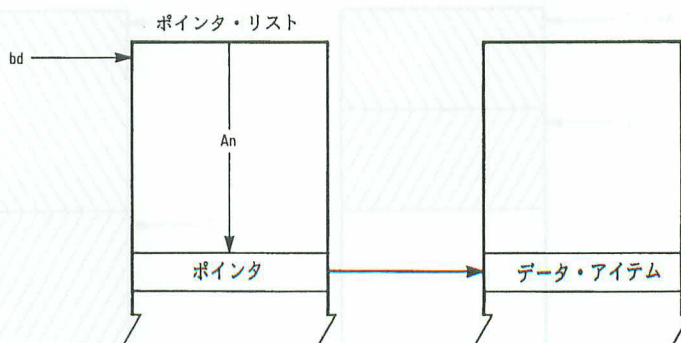



図2-9　間接アドレッシング、サプレスド・インデックス・レジスタ

メモリ間接アドレッシング・モードは次の5つの基本形式によりベース・ディスプレースメントで使用されます。

1. [bd, An] ——サプレスド・インデックス・レジスタ間接
2. ([bd, An, Xn]) ——プリインデックス付き間接
3. ([bd, An], Xn) ——ポストインデックス付き間接
4. ([bd, An], od) ——プリインデックス付き間接(アウタ・ディスプレースメント)
5. ([bd, An], Xn, od) ——ポストインデックス付き間接(アウタ・ディスプレースメント)

図2-9に示すサプレス・インデックス・レジスタ間接モードは、レジスタAnの内容をディスプレースメントで指定されるアドレスにあるポインタへのインデックスとして使用します。実際のデータ・アイテムは、選択されたポインタのアドレスにあります。

図2-10に示すプリインデックス付き間接モードは、Anの内容をディスプレースメントにあるポインタ・リスト構造へのインデックスとして使用します。レジスタXnはポインタへのインデックスで、ポインタはデータ・アイテムのアドレスをもっています。

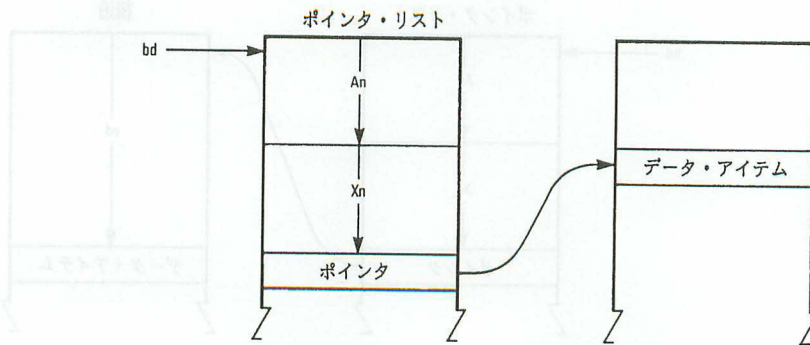



図2-10　プリインデックス付き間接アドレッシング

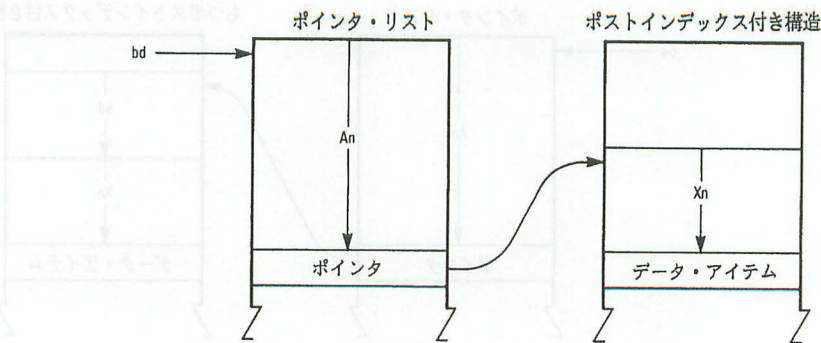



図2-11　ポストインデックス付き間接アドレッシング

図2-11に示すポストインデックス付き間接モードは、Anの内容をディスプレースメントにあるポインタ・リストへのインデックスとして使用します。レジスタXnはポインタで指定されるアドレスにあるデータ・アイテムの構造へのインデックスとして使用されます。図2-12にアウタ・ディスプレースメント・モードをもつプレインデックス付き間接アドレッシングを示します。

図2-13に示すアウタ・ディスプレースメントをもつポストインデックス付き間接モードは、Anの内容をディスプレースメントにあるポインタ・リストへのインデックスとして使用します。レジスタXnは、ポインタで指定されるアドレスにあるデータ・アイテムの構造へのインデックスとして使用されます。アウタ・ディスプレースメント(od)は、選択されたデータ構造内のデータ・アイテムのディスプレースメントです。

2.6.2　一般アドレッシング・モードの要約

前項で説明したアドレッシング・モードは、インデックス・モードのオプションの特定の組合せ、あるいは2種類のアドレッシング・モードの選択から派生したものです。たとえば、レジスタ間接(Rn)とよぶアドレッシング・モードは、レジスタがアドレス・レジスタの場合は、アドレス・レジ

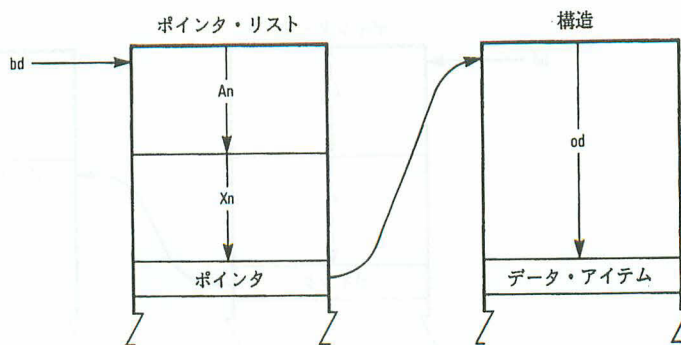



図2-12　プリインデックス付き間接(アウタ・ディスプレースメント)

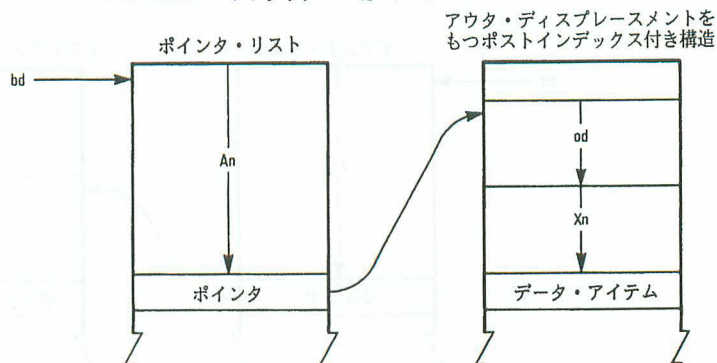



図2-13　ポストインデックス付き間接(アウタ・ディスプレースメント)

スタ間接としてアセンブルされます。Rnがデータ・レジスタの場合、アセンブラはデータ・レジスタを間接レジスタとして使用したインデックス付きアドレス・レジスタ間接モードを使用し、実効アドレスの指定でベース・サプレス・ビットをセットしてアドレス・レジスタをサプレスします。アドレス・レジスタをRnとして割り当てたほうが、データ・レジスタをRnとして使用するよりも高性能が得られます。もう1つの例は(bd, An)で、これはディスプレースメントのサイズに応じてアドレッシング・モードを選択します。ディスプレースメントが16ビット以下の場合は、ディスプレースメント付きアドレス・レジスタ間接(d16,An)モードが使用されます。32ビットのディスプレースメントが必要なときは、インデックス・レジスタをサプレスして、インデックス付きアドレス・レジスタ間接(bd, An, Xn)を使用します。

プログラマが使用できる派生アドレッシング・モード(実際に組み込まれているMC68030の実効アドレッシング・モードに関係なく)を調べておくと役立ちます。プログラマがこれらの選択にわずらわされることはありません。アセンブラが効率の高いモードを選択することができます。

以下に示す派生アドレッシング・モードでは、共通のプログラミング用語を使用しています。それらの定義は次のとおりです。

ポインタ　　——アドレスを表わすレジスタまたはメモリ内のロング・ワード値。
ベース　　——アドレスを表わすためにディスプレースメントと組み合わされるポイン

タ。

インデックス ——実効アドレス計算に加算される定数または変数値。定数インデックスはディスプレースメントです。変数インデックスは、常にその値をもつレジスタで表わされます。

disp ——ディスプレースメント、定数インデックス

サブスクリプト ——データ・レジスタまたはアドレス・レジスタを、1、2、4、または8バイト・サイズの配列要素を指定する変数インデックス・サブスクリプトとして使用します。

相対 ——プログラム・カウンタの内容から計算されるアドレス。このアドレスは位置独立型でプログラム空間にあります。psaddrを除く他のすべてのアドレスはデータ空間にあります。

addr —— 絶対アドレス

psaddr ——プログラム空間内の絶対アドレス。PC相対を除く他のすべてのアドレスがデータ空間にあります。

プリインデックス ——絶対アドレスからプログラム・カウンタ相対までのすべてのモード

ポストインデックス ——次のいずれかのモード：

addr	——データ空間での絶対アドレス
psaddr, ZPC	——プログラム空間での絶対アドレス
An	——レジスタ・ポインタ
disp, An	——定数ディスプレースメント付きレジスタ・ポインタ
addr, An	——単独変数名付き絶対アドレス
disp, PC	——単純PC相対

MC68030アーキテクチャで用意されたアドレッシング・モードから派生したプログラミング用語で定義されるアドレッシング・モードは、次のとおりです。

イミディエイト・データ——# data

命令ストリーム中の定数データ

レジスタ直接——Rn

レジスタの内容がオペランド

スキャニング・モード

(An) +	使用後アドレス・レジスタ・ポインタが自動的にインクリメントされる。
－(An)	使用前アドレス・レジスタ・ポインタが自動的にデクリメントされる。

絶対アドレス

(addr)	データ空間内の絶対アドレス
(psaddr, ZPC)	プログラム空間内の絶対アドレス。記号ZPCはPCをサプレスするが、PC相対モードを保持してプログラム空間に直接アクセスする。

レジスタ・ポインタ

(Rn)	ポインタとしてのレジスタ
(disp, Rn)	ポインタとしてのレジスタおよび定数インデックス（またはベース・アドレス）

インデックス付き

(An, Rn)	変数インデックスRn付きレジスタ・ポインタAn
(disp, An, Rn)	定数および変数インデックス付きレジスタ・ポインタ（または変数インデックス付きベース・アドレス）

(addr, Rn)　変数インデックス付き絶対アドレス
(addr, An, Rn) 2つの変数インデックス付き絶対アドレス

添字付き

－(An, Rn * scale)　アドレス・レジスタ・ポインタ添字
(disp, An, Rn * scale)　定数ディスプレースメント付きアドレス・レジスタ・ポインタ添字（または添字付きベース・アドレス）
(addr, Rn * scale)　添字付き絶対アドレス
(addr, An, Rn * scale)　変数インデックス付き絶対アドレス添字

プログラム相対

(disp, PC)　単純相対
(disp, PC, Rn)　変数インデックス付き相対
(disp, PC, Rn * scale)　添字付き相対

メモリ・ポインタ

([preindexed])　データ・オペランドを直接指すメモリ・ポインタ
([preindexed],disp)　データ・オペランドに対するディスプレースメント付きベースとしてのメモリ・ポインタ
([postindexed], Rn)　変数インデックス付きメモリ・ポインタ
([postindexed], disp, Rn)　定数および変数インデックス付きメモリ・ポインタ
([postindexed], Rn * scale)　添字付きメモリ・ポインタ
([postindexed], disp, Rn * scale)　定数インデックス添字付きメモリ・ポインタ

2. 7　M68000ファミリ間でのアドレッシングの互換性

プログラムは、M68000プロセッサ・ファミリのあるメンバから別のメンバに、上位方向に容易に移行できます。このファミリの初期メンバのユーザ・オブジェクト・コードは、新しいメンバのオブジェクト・コードと上位互換性がありますので、そのまま新しいマイクロプロセッサ上で実行することができます。アドレス拡張ワードには、MC68020/MC68030が基本M68000ファミリ・アーキテクチャに対する新しいアドレス拡張を判別できる情報がエンコードされています。初期のMC68000/008/010マイクロプロセッサ、および最新の32ビットMC68020/MC68030マイクロプロセッサのアドレス拡張ワードを図2-14に示します。MC68020/MC68030で使用するSCALEのエンコーディングは、M68000アーキテクチャの互換性を維持しながら拡張されています。SCALE値が0のときは、両方の拡張ワードが同じエンコーディングになりますので、このエンコーディングを使用するソフトウェアは、この製品系列の全プロセッサ間で上位および下位両方の互換性があります。しかし、他のSCALE値はいずれの拡張フォーマットにもありませんので、ソフトウェアは上位互換方向には容易に移行できますが、下位方向に対しては、スケールされないアドレッシングしかサポートされません。MC68000がスケーリング・ファクタをエンコードする命令を実行しようとすると、このスケーリング・ファクタは無視され、希望のメモリ・アドレスにはアクセスできなくなります。初期のマイクロプロセッサは、新しいプロセッサがインプリメントする拡張ワードに関知せず違法命令を検出するため、拡張ワードの無効エンコーディングを例外として扱うことはありません。

MC68000/MC68008/MC68010のアドレス拡張ワード

15	14 13 12	11	10	9	8	7 6 5 4 3 2 1 0
D/A	レジスタ	W/L	0	0	0	ディスプレースメント整数

D/A：0＝データ・レジスタ選択
　　　1＝アドレス・レジスタ選択
W/L：0＝ワード・サイズ操作
　　　1＝ロング・ワード・サイズ操作

MC68020/MC68030のアドレス拡張ワード

15	14 13 12	11	10 9	8	7 6 5 4 3 2 1 0
D/A	レジスタ	W/L	スケール	0	ディスプレースメント整数

D/A：0＝データ・レジスタ選択
　　　1＝アドレス・レジスタ選択
W/L：0＝ワード・サイズ操作
　　　1＝ロング・ワード・サイズ操作

スケール：
00＝スケール・ファクタ1（MC68000とコンパチブル）
01＝スケール・ファクタ2（MC68000への拡張）
10＝スケール・ファクタ4（MC68000への拡張）
11＝スケール・ファクタ8（MC68000への拡張）

図2-14　M68000ファミリのアドレス拡張ワード

2.8　その他のデータ構造

スタックとキューは広範に使用されるデータ構造です。MC68030はシステム・スタックをインプリメントし、ユーザ・スタックとキューの使用をサポートする命令も用意しています。

2.8.1　システム・スタック

アドレス・レジスタ7(A7)は、システム・スタック・ポインタ(SP)として使用されます。任意の時点で3つのシステム・スタック・ポインタのうち、いずれか1つがアクティブになっています。ステータス・レジスタのMビットおよびSビットによって、どのスタック・ポインタを使用するか否かが決まります。S＝0でユーザ・モード(ユーザ特権レベル)を示しているときには、ユーザ・スタック・ポインタ(USP)がアクティブ・システム・ポインタであり、マスタおよび割込みスタック・ポインタは参照できません。S＝1でスーパバイザ・モード(スーパバイザ特権レベル)を示していて、かつM＝1の場合、マスタ・スタック・ポインタ(MSP)がアクティブ・システム・ポインタです。S＝1およびM＝0のときには、割込みスタック・ポインタ(ISP)がアクティブ・システム・スタック・ポインタになります。このモードはMC68030のリセット後のデフォルト・モードであり、MC68000、MC68008、MC68010のスーパバイザ・モードに対応します。スーパバイザ・スタック・ポインタ(SSP)という用語は、Mビットの状態によってマスタまたは割込みスタック・ポインタを指します。M＝1のとき、SSP(またはA7)はMSPアドレス・レジスタを指します。M＝0のときSSP(またはA7)はISPアドレス・レジスタを指します。アクティブなシステム・スタック・ポインタは、システム・スタックを使用するすべての命令によって暗黙に参照されます。各システム・スタックはメモリの上位アドレスから下位アドレスに向かって使用されます。

プログラム・カウンタは、サブルーチン・コールによってアクティブ・システム・スタックにセーブされ、リターン時にアクティブ・システム・スタックから復元されます。トラップや割込みを処理している間、プログラム・カウンタとステータス・レジスタは両方ともスーパバイザ・スタック(マスタまたは割込みのいずれか)にセーブされています。したがって、スーパバイザ・レベルでのプログラムの実行は、ユーザ・プログラムの動作やユーザ・スタックの状態には関係ないため、ユー

ザ・プログラムはスーパバイザ・スタックの要求条件には関係なくユーザ・スタック・ポインタを使用することができます。

処理効率が最大になるようにシステム・スタックのデータを整列させておくために、アクティブなスタック・ポインタは、スタックとの間で転送されるすべてのバイト・サイズのオペランドに対し2だけデクリメントまたはインクリメントされます。ロング・ワード構成のメモリでは、スタック・ポインタをロング・ワード・アドレスに整列させると、例外フレーム、サブルーチン・コールおよびリターンのスタッキングや他のスタッキング操作の効率を大幅に向上させることができます。

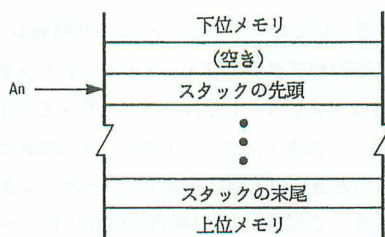
2. 8. 2 ユーザ・プログラム・スタック

ユーザは、ポストインクリメント付きおよびプリデクリメント付きのアドレス・レジスタ間接アドレッシング・モードを使用して、いくつかのスタックを実現することができます。アドレス・レジスタAn(n＝0～6)により、ユーザはメモリの上位アドレスから下位アドレス方向、あるいはその逆方向に使用されるスタックを実現することができます。この場合、次のような点に注意してください。

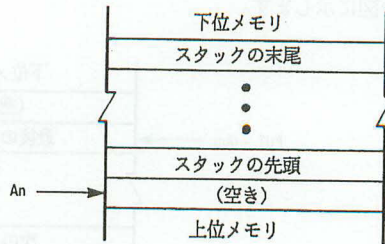
- レジスタをデクリメントしてから、その内容をスタックへのポインタとして使用する場合は、プリデクリメント・モードを使用します。
- レジスタをインクリメントしてから、その内容をスタックへのポインタとして使用する場合は、ポストインクリメント・モードを使用します。
- これらのスタックでバイト、ワード、およびロング・ワードのアイテムを混在させて使用するときは、スタック・ポインタを正しく維持してください。

メモリの上位アドレスから下位アドレス方向にデータを追加するスタックは、次の操作によって実現されます。

　　－(An)　でデータをスタックにプッシュする。
　　(An)＋　でデータをスタックからプルする。

このタイプのスタックでは、プッシュまたはプル操作が行なわれた後、レジスタAnがスタックの先頭のアイテムを指しています。この様子を次の図に示します。

メモリの下位アドレスから上位アドレス方向にデータを追加するスタックは、次の操作によって実現されます。

　　(An)＋　でデータをスタックにプッシュする。
　　－(An)　でデータをスタックからプルする。

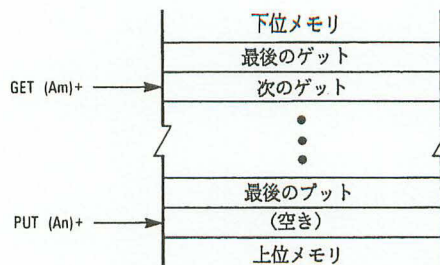
プッシュまたはプルの操作の後、レジスタAnはスタック上の次に使用する場所を指しています。この様子を次の図に示します。

2.8.3　キュー

ユーザは、ポストインクリメント付きまたはプリデクリメント付きアドレス・レジスタ間接アドレッシング・モードを使用して、キューを実現することができます。1組のアドレス・レジスタ(A0～A6のうちの2つ)を使用して、上位アドレスから下位アドレス、または下位アドレスから上位アドレス方向に使用されるキューを実現することができます。キューは一方の端からプッシュされ、他の端からプルされるので、2つのレジスタを使用します。そのうちの1つのレジスタAnには、“プット”ポインタがあり、もう1つのレジスタAmには“ゲット”ポインタがあります。

下位アドレスから上位アドレス方向に使用されるキューは、次の操作によって実現されます。

(An)＋　でデータをキューにプットする。

(Am)＋　でデータをキューからゲットする。

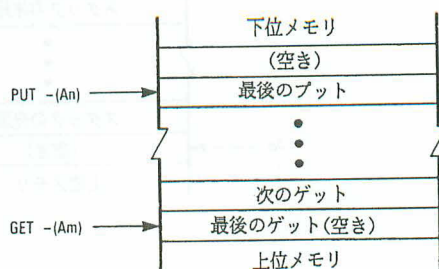
“プット”操作の後、“プット”アドレス・レジスタはキューで次に利用可能な場所を指し、“ゲット”アドレス・レジスタの内容は不変で、キューから次に取り出すアイテムを指しています。“ゲット”操作の後、“ゲット”アドレス・レジスタは、キューから次に取り出すアイテムを指し、“プット”アドレス・レジスタの内容は不変で、キューで次に利用可能な場所を指しています。この様子を次の図に示します。

このキューを循環バッファとして実現するには、関連のアドレス・レジスタをチェックし、必要に応じて“プット”または“ゲット”操作を実行する前に調整しなければなりません。アドレス・レジスタは、それからバッファの長さ(バイト単位)を差し引くことによって調整されます。

メモリの上位アドレスから下位アドレス方向に使用されるキューは、次の操作によって実現されます。

－(An)　でデータをキューにプットする。

－(Am)　でデータをキューからゲットする。

“プット”操作の後、“プット”アドレス・レジスタはキューに最後に入れられたアイテムを指し、“ゲット”アドレス・レジスタは不変で、キューから最後に取り出されたアイテムの場所を指します。“ゲット”操作の後、“ゲット”アドレス・レジスタはキューから最後に取り出されたアイテムの場所を指し、“プット”アドレス・レジスタは不変でキューに最後に入れられたアイテムを指します。

この様子を次の図に示します。

このキューを循環バッファとして実現するには、最初に“ゲット”または“プット”操作を実行してから、関連のアドレス・レジスタをチェックし、必要に応じて調整しなければなりません。アドレス・レジスタはレジスタの内容にバッファ長さ(バイト単位)を加算して調整されます。

第3章

命令セット

本章ではMC68030マイクロプロセッサの命令セットについて説明し、それらの使用法を示します。この中には、命令フォーマットと命令で使用するオペランドも含まれます。まずカテゴリ別に命令セットを分類しその概要を述べます。次に、アルファベット順に、各命令の詳細な説明、プログラミング情報、コンディション・コードの計算、および命令フォーマットの要約について解説します。

3.1　命令のフォーマット

命令の長さは、図3-1に示すように最低1ワードから11ワードまであります。命令の長さと実行する操作は命令の第1ワード、すなわちオペレーション・ワードで指定されます。残りのワードを拡張ワードとよび、これによってその命令の詳細とオペランドを指定します。これらのワードは、イミディエイト・オペランド、オペレーション・ワードで指定される実効アドレス・モードに対する拡張、分岐ディスプレースメント、ビット番号またはビット・フィールド指定、特殊レジスタ指定、トラップ・オペランド、パック/アンパック定数、引数の個数、またはコプロセッサのコンディション・コードなどに使用できます。

実行する機能を指定するオペレーション・コードのほかに、命令はその機能の対象となるオペランドのロケーションを定義します。命令はオペランドのロケーションを次の3とおりの方法で指定します。

- レジスタ指定――命令のレジスタ・フィールドがレジスタ番号をもっている場合。
- 実効アドレス――命令の実効アドレス・フィールドにアドレス・モード情報が含まれている場合。
- インプリシット参照――命令の機能の定義の中で特定のレジスタを使用することが決められている場合。

0　　　　　　　　　　　　　　　　　　　　　　　　　　　　15
オペレーション・ワード (1ワード、操作およびモードを指定)
特殊オペランド指定子 (もしあれば、1または2ワード)
イミディエイト・オペランドまたはソース実効アドレスの拡張 (もしあれば、1～5ワード)
デスティネーション実効アドレスの拡張 (もしあれば、1～5ワード)

図3-1　命令ワードの一般フォーマット

命令のレジスタ・フィールドは、使用するレジスタを指定します。命令の他のフィールドは、選択されたレジスタがアドレス・レジスタかデータ・レジスタかということと、そのレジスタの使用法を指定します。第2章にレジスタに関する詳細な情報を記載しています。

実効アドレス情報には、レジスタ、ディスプレースメント、およびその実効アドレス・モードに対する絶対アドレスが含まれています。第2章に実効アドレス・モードの詳細な説明があります。

一部の命令は特定のレジスタに対してのみ操作を実行します。これらの命令は必要なレジスタを暗黙に指定します。

3.2　命令の概要

MC68030には次の操作を実行する命令があります。

- データ転送
- 整数算術演算
- 論理演算
- シフトおよびローテイト
- ビット操作
- ビット・フィールド操作
- 2進化10進演算
- プログラム制御
- システム制御
- マルチプロセッサ通信

全範囲にわたる命令の機能と、前述した多様なアドレッシング・モードを組み合わせることによって、柔軟性に富んだプログラム開発を行なうことができます。

本章では、次の表記法を使用しています。命令記述のオペランド・シンタックス・ステートメントでは、右側のオペランドがデスティネーション・オペランドになります。

An＝アドレス・レジスタ、A0〜A7
Dn＝データ・レジスタ、D0〜D7
Rn＝アドレス・レジスタまたはデータ・レジスタ
CCR＝コンディション・コード・レジスタ（ステータス・レジスタの下位バイト）
cc＝CCRからのコンディション・コード
SR＝ステータス・レジスタ
cpcc＝コプロセッサ・コンディション・コード
SP＝アクティブ・スタック・ポインタ
USP＝ユーザ・スタック・ポインタ
ISP＝スーパバイザ/割込みスタック・ポインタ
MSP＝スーパバイザ/マスタ・スタック・ポインタ
SSP＝スーパバイザ（マスタまたは割込み）スタック・ポインタ
DFC＝デスティネーション・ファンクション・コード・レジスタ
SFC＝ソース・ファンクション・コード・レジスタ
Rc＝制御レジスタ（VBR, SFC, DFC, CACR, CAAR）
MRc＝MMUレジスタ（SRP, CRP, TC, TT0, TT1）
MMUSR＝MMUステータス・レジスタ
d＝ディスプレースメント、d_{16}は16ビットのディスプレースメント
<ea>＝実効アドレス
list＝レジスタのリスト、たとえば、D0〜D3
#<data>＝イミディエイト・データ；リテラル整数
{offset：width}＝ビット・フィールド選択
label＝アセンブリ・プログラムのラベル

[7] ＝オペランドのビット7
[31：24] ＝オペランドのビット31～24(レジスタの最上位バイト)
X＝CCRの拡張(X)ビット
N＝CCRの負(N)ビット
Z＝CCRのゼロ(Z)ビット
V＝CCRのオーバフロー(V)ビット
C＝CCRのキャリ(C)ビット
＋＝算術加算またはポストインクリメント指示子
－＝算術減算またはプリデクリメント指示子
＊＝算術乗算
/＝算術除算または結合記号
～＝インバート、オペランドが論理的に補数化されることを示す。
Λ＝論理AND
V＝論理的OR
⊕＝排他的論理OR
Dc＝比較に使用されるデータ・レジスタ、D0～D7
Du＝更新に使用されるデータ・レジスタ、D0～D7
Dr, Dq＝データ・レジスタ、除算の余りまたは商
Dh, Dl＝データ・レジスタ、乗算結果の上位32ビットおよび下位32ビット
MSW＝最上位ワード
LSW＝最下位ワード
MSB＝最上位ビット
FC＝ファンクション・コード
{R/W} ＝リードまたはライト指示子
[An] ＝アドレス拡張

3.2.1　データ転送命令

基本的にMOVE命令とそれに関連するアドレッシング・モードによって、アドレスとデータの転送および格納操作を実行します。MOVE命令は、バイト、ワード、およびロング・ワードのオペランドを、メモリからメモリ、メモリからレジスタ、レジスタからメモリ、そしてレジスタからレジスタへ転送します。アドレス転送命令(MOVEまたはMOVEA)は、ワードおよびロング・ワード・オペランドを転送し、有効なアドレス操作だけを実行します。また、一般MOVE命令のほかに、次のような特殊データ転送命令があります。

MOVEM(複数レジスタ転送)
MOVEP(周辺データ転送)
MOVEQ(クイック転送)
EXG(レジスタ交換)
LEA(実効アドレスのロード)
PEA(実効アドレスのプッシュ)
LINK(スタックのリンク)
UNLK(スタックのリンク解除)

表3-1にデータ転送操作の概要を示します。

表3-1　データ転送操作

命　　令	オペランド・シンタックス	オペランド・サイズ	操　　作
EXG	Rn, Rn	32	Rn↔Rn
LEA	〈ea〉, An	32	〈ea〉→An
LINK	An, #〈d〉	16, 32	Sp－4→SP；An→(SP)；SP→An；SP＋d→SP
MOVE MOVEA	〈ea〉, 〈ea〉 〈ea〉, An	8, 16, 32 16, 32→32	ソース→デスティネーション
MOVEM	list, 〈ea〉 〈ea〉, list	16, 32 16, 32→32	リストされたレジスタ→デスティネーション ソース→リストされたレジスタ
MOVEP	Dn, (d_{16}, An) (d_{16}, An),Dn	16, 32	Dn [31：24] → (An＋d)；Dn [23：16] → (An＋d＋2)； Dn [15：8] → (An＋d＋4)；Dn [7：0] → (An＋d＋6) (An＋d) →Dn [31：24]；(An＋d＋2) →Dn [23：16]； (An＋d＋4) →Dn [15：8]；(An＋d＋6) →Dn [7：0]
MOVEQ	#〈data〉, Dn	8→32	イミディエイト・データ→デスティネーション
PEA	〈ea〉	32	Sp－4→SP；〈ea〉→(SP)
UNLK	An	32	An→SP；(SP)→An；SP＋4→SP

3. 2. 2　整数算術演算命令

算術演算操作には、加算(ADD)、減算(SUB)、乗算(MUL)、および除算(DIV)の4つの基本操作のほかに、算術比較(CMP、CMPM、CMP2)、クリア(CLR)、およびネゲート(NEG)があります。ADD、CMP、およびSUBの各命令はアドレス操作とデータ操作の両方に使用でき、データ操作に対してはすべてのオペランド・サイズが有効です。アドレス・オペランドは、16ビットまたは32ビットで構成されます。クリアおよびネゲート命令はすべてのサイズのデータ・オペランドを扱うことができます。

符号付きおよび符号なしMULおよびDIV命令には次のものがあります。

- ●ロング・ワードの積を生成するワード乗算
- ●ロング・ワードまたはクワッド・ワードの積を生成するロング・ワード乗算
- ●ロング・ワード被除算をワード除数で除算(ワードの商とワードの余り)
- ●ロング・ワードまたはクワッド・ワードの被除数をロング・ワードの除数で除算(ロング・ワードの商とロング・ワードの余り)

拡張命令を組み合わせて使用すれば、多倍精度や異種サイズの算術演算を実行することができます。拡張命令には次のものがあります。

ADDX(拡張加算)	EXT(符号拡張)
SUBX(拡張減算)	NEGX(拡張ネゲート)

表3-2に整数算術操作の概要を示します。

3. 2. 3　論理操作

論理操作命令(AND、OR、EOR、およびNOT)は、すべてのサイズの整数データ・オペランドについて論理操作を実行します。また、類似のイミディエイト命令(ANDI、ORI、およびEORI) はすべてのサイズのイミディエイト・データについて、これらの論理操作を実行します。TST命令はオペランドと0との算術比較を行ない、その結果をコンディション・コードに反映します。

表3-3に論理操作の概要を示します。

表3-2　整数算術演算操作

命　令	オペランド・シンタックス	オペランド・サイズ	操　作
ADD ADDA	Dn,〈ea〉 〈ea〉, Dn 〈ea〉, An	8, 16, 32 8, 16, 32 16, 32	ソース+デスティネーション→デスティネーション
ADDI ADDQ	#〈data〉,〈ea〉 #〈data〉,〈ea〉	8, 16, 32 8, 16, 32	イミディエイト・データ+デスティネーション→デスティネーション
ADDX	Dn, Dn −(An), −(An)	8, 16, 32 8, 16, 32	ソース+デスティネーション+X→デスティネーション
CLR	〈ea〉	8, 16, 32	0→デスティネーション
CMP CMPA	〈ea〉,Dn 〈ea〉, An	8, 16, 32 16, 32	デスティネーション−ソース
CMP1	#〈data〉,〈ea〉	8, 16, 32	デスティネーション−イミディエイト・データ
CMPM	(An)+, (An)+	8, 16, 32	デスティネーション−ソース
CMP2	〈ea〉, Rn	8, 16, 32	下限<=Rn<=上限
DIVS/DIVU DIVSL/DIVUL	〈ea〉, Dn 〈ea〉, Dr : Dq 〈ea〉, Dq 〈ea〉, Dr : Dq	32/16→16 : 16 64/32→32 : 32 32/32→32 32/32→32 : 32	デスティネーション/ソース→デスティネーション (符号付きまたは符号なし)
EXT EXTB	Dn Dn Dn	8→16 16→32 8→32	符号拡張デスティネーション→デスティネーション
MULS/MULU	〈ea〉, Dn 〈ea〉, Dl 〈ea〉, Dh : Dl	16×16→32 32×32→32 32×32→64	ソース'デスティネーション→デスティネーション (符号付きまたは符号なし)
NEG	〈ea〉	8, 16, 32	0−デスティネーション→デスティネーション
NEGX	〈ea〉	8, 16, 32	0−デスティネーション−X→デスティネーション
SUB SUBA	〈ea〉, Dn Dn,〈ea〉 〈ea〉, An	8, 16, 32 8, 16, 32 16, 32	デスティネーション−ソース→デスティネーション
SUBI SUBQ	#〈data〉,〈ea〉 #〈data〉,〈ea〉	8, 16, 32 8, 16, 32	デスティネーション−イミディエイト・データ→デスティネーション
SUBX	Dn, Dn −(An), −(An)	8, 16, 32 8, 16, 32	デスティネーション−ソース−X→デスティネーション

表3-3　論理操作

命　令	オペランド・シンタックス	オペランド・サイズ	操　作
AND	〈ea〉, Dn Dn,〈ea〉	8, 16, 32 8, 16, 32	ソースΛデスティネーション→デスティネーション
ANDI	#〈data〉,〈ea〉	8, 16, 32	イミディエイト・データΛデスティネーション→デスティネーション
EOR	Dn,〈ea〉	8, 16, 32	ソース⊕デスティネーション→デスティネーション
EORI	#〈data〉,〈ea〉	8, 16, 32	イミディエイト・データ⊕デスティネーション→デスティネーション
NOT	〈ea〉	8, 16, 32	～デスティネーション→デスティネーション
OR	〈ea〉, Dn Dn,〈ea〉	8, 16, 32 8, 16, 32	ソースVデスティネーション→デスティネーション
ORI	#〈data〉,〈ea〉	8, 16, 32	イミディエイト・データVデスティネーション→デスティネーション
TST	〈ea〉	8, 16, 32	ソース−0でコンディション・コードをセット

表3-4　シフトおよびローテイト操作

命　令	オペランド・シンタックス	オペランド・サイズ	操　作
ASL	Dn, Dn # 〈data〉, Dn 〈ea〉	8, 16, 32 8, 16, 32 16	X/C ← [←] ← 0
ASR	Dn, Dn # 〈data〉, Dn 〈ea〉	8, 16, 32 8, 16, 32 16	[→] → X/C
LSL	Dn, Dn # 〈data〉, Dn 〈ea〉	8, 16, 32 8, 16, 32 16	X/C ← [←] ← 0
LSR	Dn, Dn # 〈data〉, Dn 〈ea〉	8, 16, 32 8, 16, 32 16	0 → [→] → X/C
ROL	Dn, Dn # 〈data〉, Dn 〈ea〉	8, 16, 32 8, 16, 32 16	C ← [←]
ROR	Dn, Dn # 〈data〉, Dn 〈ea〉	8, 16, 32 8, 16, 32 16	[→] → C
ROXL	Dn, Dn # 〈data〉, Dn 〈ea〉	8, 16, 32 8, 16, 32 16	C ← [←] ← X
ROXR	Dn, Dn # 〈data〉, Dn 〈ea〉	8, 16, 32 8, 16, 32 16	X → [→] → C
SWAP	Dn	32	MSW ⇄ LSW

3.2.4　シフトおよびローテイト命令

算術シフト命令ASR、ASL、および論理シフト命令LSR、LSLは両方向へのシフト操作を実行します。ROR、ROL、ROXR、およびROXL命令は、拡張ビット付きおよび拡張ビットなしでローテイト(循環シフト)操作を実行します。シフトおよびローテイト操作はすべて、レジスタまたはメモリのいずれに対しても実行することができます。

レジスタのシフトおよびローテイト操作は、すべてのオペランド・サイズのシフトを行なうことができます。シフト数は命令のオペレーション・ワード(1～8ビットのシフト)またはレジスタ(モジュロ64のシフト・カウント)で指定することができます。

メモリのシフトおよびローテイト操作は1ワード長のオペランドを1ビットだけシフトします。SWAP命令はレジスタの上位16ビットと下位16ビットを交換します。シフト/ローテイト命令の性能が強化されているため、シフト回数に8を指定したRORまたはROL命令を使用することにより、高速なバイト交換が可能です。表3-4にシフトおよびローテイト操作の概要を示します。

3.2.5　ビット操作命令

ビット操作は、ビット・テスト(BTST)、ビット・テストおよびセット(BSET)、ビット・テスト

表3-5　ビット操作

命　　令	オペランド・シンタックス	オペランド・サイズ	操　　作
BCHG	Dn, 〈ea〉 # 〈data〉, 〈ea〉	8, 32 8, 32	～（デスティネーションの<ビット番号>）→Z→ デスティネーションのビット
BCLR	Dn, 〈ea〉 # 〈data〉, 〈ea〉	8, 32 8, 32	～（デスティネーションの<ビット番号>）→Z； 0→デスティネーションのビット
BSET	Dn, 〈ea〉 # 〈data〉, 〈ea〉	8, 32 8, 32	～（デスティネーションの<ビット番号>）→Z； 1→デスティネーションのビット
BTST	Dn, 〈ea〉 # 〈data〉, 〈ea〉	8, 32 8, 32	～（デスティネーションの<ビット番号>）→Z

表3-6　ビット・フィールド操作

命　　令	オペランド・シンタックス	オペランド・サイズ	操　　作
BFCHG	〈ea〉{offset : width}	1－32	～フィールド→フィールド
BFCLR	〈ea〉{offset : width}	1－32	0's→フィールド
BFEXTS	〈ea〉{offset : width}, Dn	1－32	フィールド→Dn；符号付き拡張
BFEXTU	〈ea〉{offset : width}, Dn	1－32	フィールド→Dn；ゼロ拡張
BFFFO	〈ea〉{offset : width}, Dn	1－32	フィールドでセットされている最初のビットをスキャン；オフセット→Dn
BFINS	Dn, 〈ea〉{offset : width}	1－32	Dn→フィールド
BFSET	〈ea〉{offset : width}	1－32	1's→フィールド
BFTST	〈ea〉{offset : width}	1－32	フィールドMSB→N；～（フィールド内の全ビットのOR）→Z

注：ビット・フィールド命令はすべて、BFTST命令に示すようにNおよびZビットをセットしたあと指定された操作を実行します。

およびクリア(BCLR)、そしてビット・テストおよび変更(BCHG)の各命令を使用して実行されます。すべてのビット操作はレジスタまたはメモリのいずれに対しても実行できます。ビット番号はイミディエイト・データとして、あるいはデータ・レジスタで指定されます。レジスタ・オペランドは32ビット長で、メモリ・オペランドは8ビット長です。表3-5にビット操作の概要を示します。Zはビット2、つまりステータス・レジスタの"ゼロ"ビットです。

3.2.6　ビット・フィールド命令

MC68030は32ビットまでのフィールドに対する可変長ビット・フィールド操作をサポートしています。ビット・フィールド挿入(BFINS)命令は、ビット・フィールドに値を挿入します。

ビット・フィールド符号なし抽出(BFEXTU)命令およびビット・フィールド符号付き抽出(BFEXTS)命令は、フィールドから値を抽出します。ビット・フィールド・ファインド・ファースト・ワン(BFFFO)は、ビット・フィールドの中でセットされている最初のビットを見つけます。また、ビット操作命令に類似した命令、すなわちビット・フィールド・テスト(BFTST)、ビット・フィールド・テストおよびセット(BFSET)、ビット・フィールド・テストおよびクリア(BFCLR)、そしてビット・フィールド・テストおよび変更(BFCHG)の各命令も含まれています。表3-6にビット・フィールド操作の概要を示します。

3.2.7　2進化10進演算命令

5つの命令が2進化10進数の演算をサポートしています。パック2進化10進数に対する算術演算は、拡張10進加算(ABCD)、拡張10進減算(SBCD)、拡張10進ネゲート(NBCD)の各命令で実行されます。PACKおよびUNPACK命令は、ASCIIやEBCDIC文字列などのバイト符号化数値データをBCDデータ、あるいはその逆に変換するのに役立ちます。表3-7に2進化10進演算の概要を示します。

表3-7　2進化10進演算操作

命　令	オペランド・シンタックス	オペランド・サイズ	操　作
ABCD	Dn, Dn −(An), −(An)	8 8	ソース$_{10}$＋デスティネーション$_{10}$＋X→デスティネーション
NBCD	〈ea〉	8	0＃デスティネーション$_{10}$−X→デスティネーション
PACK	−(An), −(An) ＃〈data〉 Dn, Dn, ＃〈data〉	16→8 16→8	アンパック・ソース＋イミディエイト・データ→ パック・デスティネーション
SBCD	Dn, Dn −(An), −(An)	8 8	デスティネーション$_{10}$−ソース$_{10}$−X→デスティネーション
UNPK	−(An), −(An) ＃〈data〉 Dn, Dn, ＃〈data〉	8→16 8→16	パック・ソース→アンパック・ソース アンパック・ソース＋イミディエイト・データ→ アンパック・デスティネーション

表3-8　プログラム制御操作

命　令	オペランド・シンタックス	オペランド・サイズ	操　作
条件付き分岐命令			
Bcc	〈label〉	8, 16, 32	条件が真の場合、PC＋d→PC
DBcc	Dn, 〈label〉	16	条件が偽の場合、Dn−1→Dn Dn≠−1の場合、PC＋d→PC
Scc	〈ea〉	8	条件が真の場合、1's→デスティネーション それ以外は、0's→デスティネーション
無条件分岐命令			
BRA	〈label〉	8, 16, 32	PC＋d→PC
BSR	〈label〉	8, 16, 32	SP−4→SP；PC→(SP)；PC＋d→PC
JMP	〈ea〉	none	デスティネーション→PC
JSR	〈ea〉	none	SP−4→SP；PC→(SP)；デスティネーション→PC
NOP	none	none	PC＋2→PC
リターン命令			
RTD	＃〈d〉	16	(SP)→PC；SP＋4＋d→SP
RTR	none	none	(SP)→CCR；SP＋2→SP；(SP)→PC；SP＋4→SP
RTS	none	none	(SP)→PC；SP＋4→SP

3. 2. 8　プログラム制御命令

プログラム制御操作は、サブルーチン・コールとリターン命令の組合せ、そして条件付きおよび無条件分岐命令によって実行されます。表3-8 にこれらの命令の概要を示します。

命令のニーモニック・オペコードのccは、次のコンディション・コードの1つをテストすることを指定します。

CC ──キャリ・クリア
CS ──キャリ・セット
EQ ──等しい
F ──真でない*
GE ──大きいか等しい
GT ──より大きい
HI ──ハイ
LE ──小さいか等しい
LS ──ローか同じ
LT ──より小さい
MI ──マイナス
NE ──等しくない
PL ──プラス
T ──常に真*

VC ──オーバフロー・クリア　　VS ──オーバフロー・セット
＊Bccまたは cpBcc命令には適用されません。

3.2.9 システム制御命令

システム制御操作は、特権命令、トラップ発生命令、およびコンディション・コード・レジスタを使用または変更する命令により実行されます。表3-9 にこれらの命令の概要を示します。前述し

表3-9　システム制御操作

命令	オペランド・シンタックス	オペランド・サイズ	操作
		特権命令	
ANDI	#〈data〉, SR	16	イミディエイト・データ∧SR→SR
EORI	#〈data〉, SR	16	イミディエイト・データ⊕SR→SR
MOVE	〈ea〉, SR SR, 〈ea〉	16 16	ソース→SR SR→デスティネーション
MOVE	USP, An An, USP	32 32	USP→An An→USP
MOVEC	Rc, Rn Rn, Rc	32 32	Rc→Rn Rn→Rc
MOVES	Rn, 〈ea〉 〈ea〉, Rn	8, 16, 32	Rn→DFCで指定されるデスティネーション SFCで指定されるソース→Rn
ORI	#〈data〉, SR	16	イミディエイト・データ∨ SR→SR
RESET	none	none	$\overline{\text{RESET}}$ラインをアサート
RTE	none	none	(SP) →SP；SP＋2→SP；(SP) →PC；SP＋4→SP； フォーマットに従ってスタックをリストア
STOP	#〈data〉	16	イミディエイト・データ→SR；STOP
		トラップ発生命令	
BKPT	#〈data〉	none	ブレーク・ポイント・サイクルが承認された場合は、返されたオペレーション・ワードを実行し、それ以外は不当命令としてトラップする。
CHK	〈ea〉, Dn	16, 32	Dn＜0またはDn＞(ea)の場合、CHK例外を発生する。
CHK2	〈ea〉, Rn	8, 16, 32	Rn＜下限またはRn＞上限の場合、CHK例外を発生する。
ILLEGAL	none	none	SSP－2→SSP；ベクタ・オフセット→ (SSP)； SSP－4→SSP；PC→ (SSP)； SSP－2→SSP；SR→ (SSP)； 違法命令ベクタ・アドレス→PC
TRAP	#〈data〉	none	SSP－2→SSP；フォーマットおよびベクタ・オフセット→ (SSP) SSP－4→SSP；PC→ (SSP)；SSP－2→SSP； SR→ (SSP)；ベクタ・アドレス→PC
TRAPcc	none #〈data〉	none 16, 32	ccが真の場合、TRAP例外を発生する。
TRAPV	none	none	Vがセットされている場合、オーバフローTRAP例外を発生する。
		コンディション・コード・レジスタ	
ANDI	#〈data〉, CCR	8	イミディエイト・データ∧CCR→CCR
EORI	#〈data〉, CCR	8	イミディエイト・データ⊕CCR→CCR
MOVE	〈ea〉, CCR CCR, 〈ea〉	16 16	ソース→CCR CCR→デスティネーション
ORI	#〈data〉, CCR	8	イミディエイト・データ∨ CCR→CCR

表3-10　メモリ管理ユニットの命令

命　令	オペランド・シンタックス	オペランド・サイズ	操　作
PFLUSHA	none	none	すべてのATCエントリを無効にする。
PFLUSH	〈FC〉, # 〈mask〉 [. 〈ea〉]	none	実効アドレスにあるすべてのATCエントリを無効にする。
PLOAD	〈FC〉, 〈ea〉, {R/W}	none	実効アドレスに対してATCエントリを生成する。
PMOVE	Rn, 〈ea〉 〈ea〉, Rn	16, 32 16, 32	レジスタn→デスティネーション ソース→レジスタn
PTEST	〈FC〉, 〈ea〉, # 〈level〉 {R/W} [. An]	none	論理アドレスに関する情報→PMMUステータス

表3-11　マルチプロセッサ操作命令

命　令	オペランド・シンタックス	オペランド・サイズ	操　作
リード・モディファイ・ライト			
CAS	Dc, Du, 〈ea〉	8, 16, 32	デスティネーション－Dc→CC;Zがセットされている場合、Du→デスティネーション そうでない場合、デスティネーション→Dc
CAS2	Dc1 : Dc2, Du1 : Du2, (Rn) : (Rn)	16, 32	デュアル・オペランドCAS
TAS	〈ea〉	8	デスティネーション－0;コンディション・コードをセット;1→デスティネーション[7]
コプロセッサ			
cpBcc	〈label〉	16, 32	cpccが真の場合、PC＋d→PC
cpDBcc	〈label〉, Dn	16	cpccが偽の場合、Dn－1→Dn Dn≠－1の場合、PC＋d→PC
cpGEN	User Defined	User Defined	オペランド→コプロセッサ
cpRESTORE	〈ea〉	none	＜ ea ＞からコプロセッサの状態を回復する。
cpSAVE	〈ea〉	none	＜ ea ＞にコプロセッサの状態をセーブする。
cpScc	〈ea〉	8	cpccが真の場合、1's→デスティネーション;そうでない場合、0's→デスティネーション
cpTRAPcc	none # 〈data〉	none 16, 32	cpccが真の場合はTRAPcc例外が発生

たコンディション・コード表記のリストはTRAPcc命令に適用されます。これらのどの命令によっても、プロセッサは命令パイプをフラッシュします。

3. 2. 10　メモリ管理ユニット命令

メモリ管理命令は、アドレス変換キャッシュ(ATC)のフラッシュ、ATCへのエントリのロード、メモリ管理ユニット(MMU)制御レジスタへのロードと格納、アドレス変換テーブルのサーチの実行、MMUステータス・レジスタへの結果の格納を行ないます。表3-10にこれらの命令の概要を示します。

3. 2. 11　マルチプロセッサ命令

TAS、CASおよびCAS2命令でマルチプロセッシング・システムにおけるプロセッサの動作を調整します。これらの命令は、リード-モデファイ-ライトのバス・サイクルを使用して、中断されないメモリ更新を保証しています。コプロセッサ命令は、コプロセッサを操作します。表3-11にこれらの命令の概要を示します。

3.3　命令セットの詳細

この項では、MC68030命令セットの各命令について詳しく説明します。まず、命令を説明するための表記法とフォーマットについて述べます。次に、各命令を詳細に解説していきます。命令の説明は、命令のニーモニックごとにアルファベット順に並べてあります。

3.3.1　表記法とフォーマット

命令の説明はオペランド、サブフィールド、および修飾子、そして命令で実行される操作に対して表記規約を使用しています。アセンブラ形式の説明では、左側のオペランドがソース・オペランドで、右側のオペランドがデスティネーション・オペランドになります。3.2節に示す表記規約が適用されます。次に、命令の説明で使用する表記規約を示します。

オペランドの表記法：

PC――プログラム・カウンタ
SR――ステータス・レジスタ
V――オーバフロー・コンディション・コード
イミディエイト・データ――命令からのイミディエイト・データ
ソース――ソースの内容
デスティネーション――デスティネーションの内容
ベクタ――例外ベクタのロケーション

規約上、デスティネーション・オペランドは右側のオペランドです。

サブフィールドと修飾子の表記法：

<オペランド>の<ビット>――オペランドの1つのビットを選択する。
<ea>｛オフセット：幅｝――ビット・フィールドを選択する。
(<オペランド>)――参照したロケーションの内容
<オペランド>$_{10}$――オペランドは2進化10進数であり、演算は10進で実行する。
(<アドレス・レジスタ>)
－(アドレス・レジスタ)
(アドレス・レジスタ)＋――レジスタ間接オペレータで、オペランド・レジスタが命令オペランドのメモリ・ロケーションを指すことを示す。オプションのモード修飾子は、－、＋、(d)および(d, ix)である。
#×××または#<data>――命令ワードに続くイミディエイト・データ

2つのオペランドをもつ操作の表記法。これらの命令は<オペランド><OP><オペランド>で記述され、<OP>は次のいずれかである：

→――ソース・オペランドがデスティネーション・オペランドに転送される。
←→――2つのオペランドの内容が交換される。
＋――2つのオペランドが加算される。
－――ソース・オペランドがデスティネーション・オペランドから減算される。
＊――2つのオペランドが乗算される。
/――ソース・オペランドがデスティネーション・オペランドで除算される。
<――関係テストを行ない、ソース・オペランドがデスティネーション・オペランドより小さければ真。
>――関係テストを行ない、ソース・オペランドがデスティネーション・オペランドより大きけ

れば真。

shifted by――ソース・オペランドが2番目オペランドで指定される位置数だけシフトまたはローテイトされる。

rotated by

単一オペランド操作の表記法：

～<オペランド>――オペランドが論理的に補数化される。

<オペランド>符号拡張――オペランドが符号拡張され、上位半分の全ビットが下位半分の最上位ビットと等しくなる。

<オペランド>のテスト――オペランドは0と比較され、コンディション・コードが適宜セットされる。

その他の表記法：

TRAP――次の操作と同じ。

フォーマット/オフセット・ワード→（SSP）；SSP－2→SSP；PC→（SSP）；SSP－4→SSP；SR→（SSP）；SSP－2→SSP；（ベクタ）→PC

STOP――ストップ状態に入り、割込みを待つ。

If <条件> then <操作> else <操作>――条件をテストする。条件が真であれば、thenのあとの操作をを実行する。条件が偽で、オプションの“else”句がある場合は、“else”のあとの操作を実行する。条件が偽でオプションの“else”句がない場合は、命令はなにも実行しない。例として、Bcc命令の説明を参照のこと。

3.3.2 コンディション・コード・レジスタ

ステータス・レジスタのコンディション・コード・レジスタ部分は、次の5ビットで構成されています。

X――拡張
N――負
Z――ゼロ
V――オーバフロー
C――キャリ

最後の4ビットは、プロセッサの動作結果の状態を表わします。表3-11にそれらのビットに対する各命令の影響を示します。Xビットは多倍精度計算のためのオペランドの1つで、これを使用したときはキャリ・ビットの値がセットされます。M68000ファミリでは、プログラミング・モデルを簡単にするためにキャリ・ビット(C)と多倍精度拡張ビット(X)は別々に設けられています。表3-4に例がありますので参照してください。

プログラムおよびシステム制御命令は、これらのビットの特定の組合せを使用してプログラムとシステム・フローの制御を行ないます。表3-12にそれらのビットの組合せと定義を示します。

命令セットの記述において、コンディション・コード・レジスタは次のように表わされます。

X	N	Z	V	C

ここで、各ビットの意味は以下のとおりです。

X（拡張）

多くの算術演算でCビットの値がセットされます。そうでない場合は、影響を受けないか、特定の結果がセットされます。

N（負）

結果の最上位ビットが1であればセットされ、そうでなければクリアされます。

Z（ゼロ）

結果が0であればセットされ、そうでなければクリアされます。

V（オーバフロー）

算術演算のオーバフローが発生した場合にセットされます。これは、結果がオペランド・サイズで表現できないことを意味します。オーバフローしなかった場合は、クリアされます。

C（キャリ）

加算において、オペランドの最上位ビットからキャリが発生した場合にセットされます。また、減算でボローが発生した場合にもセットされます。それ以外の場合はクリアされます。

各コンディション・コードの変化を表わすのに次の記号を使用しています。

＊＝操作の結果に従ってセットされる。

—＝操作では変化しない。

0＝クリアされる。

1＝セットされる。

U＝操作のあと不定となる。

3.3.3　命令の説明

図3-2に命令を説明するためのフォーマットを示します。属性の行は、命令のオペランドのサイズを指定します。命令が2種類以上のサイズのオペランドを使用できる場合は、命令のニーモニックに次のサフィックスを用いることができます。

.B——バイト・オペランド

.W——ワード・オペランド

.L——ロング・ワード・オペランド

表3-12　コンディション・コードの計算(その1)

操　　作	X	N	Z	V	C	特　殊　定　義
ABCD	*	U	?	U	?	C＝10進キャリ $Z = Z \wedge \overline{Rm} \wedge \ldots \wedge \overline{R0}$
ADD, ADDI, ADDQ	*	*	*	?	?	$V = Sm \wedge Dm \wedge \overline{Rm} \vee \overline{Sm} \wedge \overline{Dm} \wedge Rm$ $C = Sm \wedge Dm \vee \overline{Rm} \wedge Dm \vee Sm \wedge \overline{Rm}$
ADDX	*	*	?	?	?	$V = Sm \wedge Dm \wedge \overline{Rm} \vee \overline{Sm} \wedge \overline{Dm} \wedge Rm$ $C = Sm \wedge Dm \vee \overline{Rm} \wedge Dm \vee Sm \wedge \overline{Rm}$ $Z = Z \wedge \overline{Rm} \wedge \ldots \wedge \overline{R0}$
AND, ANDI, EOR, EORI, MOVEQ, MOVE, OR, ORI, CLR, EXT, NOT, TAS, TST	−	*	*	0	0	
CHK	−	*	U	U	U	
CHK2, CMP2	−	U	?	U	?	$Z = (R = LB) \vee (R = UB)$ $C = (LB <= UB) \wedge (IR < LB) \vee (R > UB)) \vee (UB < LB)$ $\wedge (R > UB) \wedge (R < LB)$
SUB, SUBI, SUBQ	*	*	*	?	?	$V = \overline{Sm} \wedge Dm \wedge \overline{Rm} \vee Sm \wedge \overline{Dm} \wedge Rm$ $C = Sm \wedge \overline{Dm} \vee Rm \wedge \overline{Dm} \vee Sm \wedge Rm$
SUBX	*	*	?	?	?	$V = \overline{Sm} \wedge Dm \wedge \overline{Rm} \vee Sm \wedge \overline{Dm} \wedge Rm$ $C = Sm \wedge \overline{Dm} \vee Rm \wedge \overline{Dm} \vee Sm \wedge Rm$ $Z = Z \wedge \overline{Rm} \wedge \ldots \wedge \overline{R0}$
CAS, CAS2, CMP, CMPI, CMPM	−	*	*	?	?	$V = \overline{Sm} \wedge Dm \wedge \overline{Rm} \vee Sm \wedge \overline{Dm} \wedge Rm$ $C = Sm \wedge \overline{Dm} \vee Rm \wedge \overline{Dm} \vee Sm \wedge Rm$
DIVS, DUVI	−	*	*	?	0	V＝除算オーバフロー
MULS, MULU	−	*	*	?	0	V＝乗算オーバフロー

表3-12　コンディション・コードの計算(その2)

操　　作	X	N	Z	V	C	特　殊　定　義
SBCD, NBCD	*	U	?	U	?	C = 10進ボロー $Z = Z \land \overline{Rm} \land \ldots \land \overline{Ro}$
NEG	*	*	*	?	?	$V = Dm \land Rm$ $C = Dm \lor Rm$
NEGX	*	*	?	?	?	$V = Dm \land Rm$ $C = Dm \lor Rm$ $Z = Z \land \overline{Rm} \land \ldots \land \overline{R0}$
BTST, BCHG, BSET, BCLR	–	–	?	–	–	$Z = \overline{Dn}$
BFTST, BFCHG, BFSET, BFCLR	–	?	?	0	0	$N = Dm$ $Z = \overline{Dm} \land \overline{Dm-1} \land \ldots \land \overline{D0}$
BFEXTS, BFEXTU, BFFFO	–	?	?	0	0	$N = Sm$ $Z = \overline{Sm} \land \overline{Sm-1} \land \ldots \land \overline{S0}$
BFINS	–	?	?	0	0	$N = Dm$ $Z = \overline{Dm} \land \overline{Dm-1} \land \ldots \land \overline{D0}$
ASL	*	*	*	?	?	$V = Dm \land (\overline{Dm-1} \lor \ldots \lor \overline{Dm-r}) \lor \overline{Dm} \land (Dm-1 \lor \ldots + Dm-r)$ $C = \overline{Dm-r+1}$
ASL (R = 0)	–	*	*	0	0	
LSL, ROXL	*	*	*	0	?	$C = Dm - r + 1$
LSR (r = 0)	–	*	*	0	0	
ROXL (r = 0)	–	*	*	0	?	C = X
ROL	–	*	*	0	?	$C = Dm - r + 1$
ROL (r = 0)	–	*	*	0	0	
ASR, LSR, ROXR	*	*	*	0	?	$C = Dr - 1$
ASR, LSR (r = 0)	–	*	*	0	0	
ROXR (r = 0)	–	*	*	0	?	C = X
ROR	–	*	*	0	?	$C = Dr - 1$
ROR (r = 0)	–	*	*	0	0	

－＝影響なし
U＝不定、結果は無意味
？＝その他――特殊定義参照
＊＝一般の場合
　X = C
　N = Rm
　$Z = \overline{Rm} \land \ldots \land \overline{R0}$
Sm＝ソース・オペランド――最上位ビット
Dm＝デスティネーション・オペランド――最上位ビット

Rm＝結果のオペランド――最上位ビット
R＝テストされるレジスタ
n＝ビット番号
r＝シフト・カウント
LB＝下限
UB＝上限
∧＝論理積
V＝論理和
$\overline{Rm}$ = Rmの否定

命令名 →

ABCD

Add Decimal with Extend

操作の説明 →

操作：　ソース $_{10}$ ＋デスティネー

命令のアセンブラ・シンタックス →

アセンブラ・シンタックス：ABCD　D
ABCD －(

属性：　サイズ＝(バイト)

命令の操作内容の説明： →

説明：　ソース・オペランドを拡張
スティネーション・ロケー
は次の2つの方法によりア
1. データ・レジスタと
ります。
2. メモリとメモリ：オ
アドレッシング・モ
この演算はバイト操作に限

コンディション・コードの影響（「3.8　コンディション・コード」参照） →

コンディション・コード：

X	N	Z	V
*	U	*	U

X－キャリ・ビットと同じ
N－不定
Z－結果が0でなければク
V－不定
C－10進キャリが発生し
注：通常、コンディション
ため、このビットを使
とができます。

命令フォーマット――オペレーション・ワードのビット・パターンとフィールド、および他の命令の一部であるワードを指定します。実効アドレスの拡張部分は図示されてません。拡張部分（もしあれば）は命令で図示されている部分のあとに続きます。MOVE命令では、ソース実効アドレス拡張が最初にあり、デスティネーション実効アドレス拡張がそのあとに続きます。 →

命令フォーマット：

15	14	13	12
1	1	0	0

命令フォーマットに必要な各種フィールドの目的と許容値 →

命令フィールド：
レジスタRxフィールド―
R/M＝0　データ・レ
R/M＝1　プリデクリ
R/Mフィールド――オペ

図3-2　命令記述フォーマット

ABCD

Add Decimal with Extend・拡張付き10進加算

操作：　ソース$_{10}$＋デスティネーション$_{10}$＋X→デスティネーション

アセンブラ・シンタックス：　ABCD　Dy, Dx

ABCD −(Ay), −(Ax)

属性：　サイズ＝(バイト)

説明：　ソース・オペランドを拡張ビット(X)とともにデスティネーション・オペランドに加算し、結果をデスティネーション・ロケーションに格納します。加算は2進化10進演算で行ないます。オペランドは次の2つの方法によりアドレス指定できます。

1. データ・レジスタとデータ・レジスタ：オペランドは、命令で指定するデータ・レジスタにあります。
2. メモリとメモリ：オペランドは命令で指定するアドレス・レジスタを用いたプリデクリメント・アドレッシング・モードでアドレス指定されます。

この演算はバイト操作に限定されます。

コンディション・コード：

X	N	Z	V	C
*	U	*	U	*

X－キャリ・ビットと同じ。

N－不定

Z－結果が0でなければクリア、それ以外のときは変化しない。

V－不定

C－10進キャリが発生したらセット、それ以外のときはクリア。

注：通常、コンディション・コードのZビットは、演算を実行する前にプログラムでセットされるため、このビットを使用して多倍精度演算を終了したとき演算結果がゼロかどうかテストすることができます。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	レジスタRx			1	0	0	0	0	R/M	レジスタRy		

命令フィールド：

レジスタRxフィールド——デスティネーション・レジスタを指定。

R/M＝0　データ・レジスタを指定。

R/M＝1　プリデクリメント・アドレッシング・モードを使用するアドレス・レジスタを指定。

R/Mフィールド——オペランドのアドレッシング・モードを指定。

0－データ・レジスタとデータ・レジスタの操作。

1－メモリとメモリの操作。

レジスタRyフィールド――ソース・レジスタを指定。

R/M＝0　データ・レジスタを指定。

R/M＝1　プリデクリメント・アドレス・モードを使用するアドレス・レジスタを指定。

ADD

Add **・加算**

操作：　ソース＋デスティネーション→デスティネーション

アセンブラのシンタックス：　ADD <ea>, Dn
　ADD Dn, <ea>

属性：　サイズ＝(バイト、ワード、ロング・ワード)

説明：　ソース・オペランドをデスティネーション・オペランドに2進加算し、結果をデスティネーション・ロケーションに格納します。操作サイズは、バイト、ワード、ロング・ワードが指定できます。命令のモード・フィールドで、オペランド・サイズのほか、<ea>またはDnのどちらがソースでどちらがデスティネーションとなるかを指定します。

コンディション・コード：

X	N	Z	V	C
*	*	*	*	*

X－キャリ・ビットと同じ。
N－結果が負であればセット、それ以外のときはクリア。
Z－結果が0であればセット、それ以外のときはクリア。
V－オーバフローが発生すればセット、それ以外のときはクリア。
C－キャリが発生すればセット、それ以外のときはクリア。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
										実行アドレス					
1	1	0	1	レジスタ			Opモード			モード			レジスタ		

命令フィールド：

レジスタ・フィールド――8つのデータ・レジスタのいずれかを指定。

Opモード・フィールド：

バイト	ワード	ロング・ワード	操　作
000	001	010	<ea> ＋<Dn>→<n>
100	101	110	<Dn> ＋<ea>→<ea>

実効アドレス・フィールド――アドレッシング・モードを指定。

a. 指定されたロケーションがソース・オペランドなら、次に示すとおりすべてのアドレッシング・モードが可。

アドレッシング・モード	モード	レジスタ
Dn	000	reg. number : Dn
An*	001	reg. number : An
(An)	010	reg. number : An
(An)+	011	reg. number : An
−(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

アドレッシング・モード	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

＊ワードおよびロング・ワードのみ。

b. 指定されたロケーションがデスティネーション・オペランドの場合は、次に示すとおりメモリ可変アドレッシング・モードのみ可。

アドレッシング・モード	モード	レジスタ
Dn	−	−
An	−	−
(An)	010	reg. number : An
(An)+	011	reg. number : An
−(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

アドレッシング・モード	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	−	−
(d_{16},PC)	−	−
(d_8,PC,Xn)	−	−
(bd,PC,Xn)	−	−
([bd,PC,Xn],od)	−	−
([bd,PC],Xn,od)	−	−

注：1. デスティネーションがデータ・レジスタの場合、Dnモードを使用します。データ・レジスタに対しては、＜ea＞がデスティネーションとなるモードは無効です。

2. デスティネーションがアドレス・レジスタのときにはADDAを使用します。ソースがイミディエイト・データのときには、ADDIとADDQを使用します。ほとんどのアセンブラは自動的にこの区別を行ないます。

ADDA

Add Address・アドレス加算

操作：　ソース＋デスティネーション→デスティネーション

アセンブラ・シンタックス：　ADDA　<ea>，An

属性：　サイズ＝(ワード、ロング・ワード)

説明：　ソース・オペランドをデスティネーションのアドレス・レジスタに加算し、結果をアドレス・レジスタに格納します。操作サイズはワードまたはロング・ワードが指定できます。操作サイズに関係なく、デスティネーションのアドレス・レジスタ全体を使用します。

コンディション・コード：　影響を受けない。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	レジスタ			Opモード			実行アドレス					
										モード			レジスタ		

命令フィールド：

レジスタ・フィールド――8つのアドレス・レジスタのいずれかを指定。常にデスティネーションとなる。

Opモード・フィールド――操作サイズを指定。

011 ―ワード操作。ソース・オペランドがロング・ワードに符号拡張され、アドレス・レジスタの32ビット全体を使用して演算が実行される。

111 ―ロング・ワード操作。

実効アドレス・フィールド――ソース・オペランドを指定。次に示すとおり、すべてのアドレッシング・モードが可。

アドレッシング・モード	モード	レジスタ
Dn	000	reg. number：Dn
An	001	reg. number：An
(An)	010	reg. number：An
(An)＋	011	reg. number：An
−(An)	100	reg. number：An
(d_{16},An)	101	reg. number：An
(d_8,An,Xn)	110	reg. number：An
(bd,An,Xn)	110	reg. number：An
([bd,An,Xn],od)	110	reg. number：An
([bd,An],Xn,od)	110	reg. number：An

アドレッシング・モード	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

ADDI

Add Immediate ・イミディエイト加算

操作：　イミディエイト・データ＋デスティネーション→デスティネーション

アセンブラ・シンタックス：　ADDI #<data>, <ea>

属性：　サイズ＝(バイト、ワード、ロング・ワード)

説明：　イミディエイト・データをデスティネーション・オペランドに加算し、結果をデスティネーション・ロケーションに格納します。操作サイズは、バイト、ワード、ロング・ワードが指定できます。イミディエイト・データのサイズは操作サイズと同じです。

コンディション・コード：

X	N	Z	V	C
*	*	*	*	*

X－キャリ・ビットと同じ。

N－結果が負であればセット、それ以外のときはクリア。

Z－結果が0であればセット、それ以外のときはクリア。

V－オーバフローが発生すればセット、それ以外のときはクリア。

C－キャリが発生すればセット、それ以外のときはクリア。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	1	0	サイズ		実効アドレス モード			実効アドレス レジスタ		
ワード・データ（16ビット）								バイト・データ（8ビット）							
ロング・データ（32ビット）															

命令フィールド：

サイズ・フィールド――操作サイズを指定。

00－バイト操作

01－ワード操作

10－ロング・ワード操作

実効アドレス・フィールド――デスティネーション・オペランドを指定。次に示すとおり、データ可変アドレッシング・モードのみ可。

アドレッシング・モード	モード	レジスタ
Dn	000	reg. number : Dn
An	–	–
(An)	010	reg. number : An
(An)+	011	reg. number : An
−(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

アドレッシング・モード	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	–	–
		–
(d_{16},PC)	–	–
(d_8,PC,Xn)	–	–
(bd,PC,Xn)	–	–
([bd,PC,Xn],od)	–	–
([bd,PC],Xn,od)	–	–

イミディエイト・フィールド――（命令直後のデータ）

サイズ＝00　データはイミディエイト・ワードの下位バイト

サイズ＝01　データはイミディエイト・ワード

サイズ＝10　データは次の2つのイミディエイト・ワード

ADDQ

Add Quick・クイック加算

操作：　イミディエイト・データ＋デスティネーション→デスティネーション

アセンブラ・シンタックス：　ADDQ　#<data>, <ea>

属性：　サイズ＝(バイト、ワード、ロング・ワード)

説明：　1から8のイミディエイト値をデスティネーション・ロケーションのオペランドに加算します。操作サイズはバイト、ワード、ロング・ワードが指定できます。アドレス・レジスタの場合には、ワード、ロング・ワード操作も可能です。アドレス・レジスタへ加算する場合、コンディション・コードは変化せず、操作サイズに関係なくデスティネーションのアドレス・レジスタ全体(32ビット)が使用されます。

コンディション・コード：

X	N	Z	V	C
*	*	*	*	*

X－キャリ・ビットと同じ。

N－結果が負であればセット、それ以外のときはクリア。

Z－結果が0であればセット、それ以外のときはクリア。

V－オーバフローが発生すればセット、それ以外のときはクリア。

C－キャリが発生すればセット、それ以外のときはクリア。

デスティネーションがアドレス・レジスタの場合には、コンディション・コードは影響を受けない。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	データ			0	サイズ		実効アドレス					
										モード			レジスタ		

命令フィールド：

データ・フィールド――3ビットのイミディエイト・データ、0－7 (0は8を表わす)。

サイズ・フィールド――操作サイズを指定。

00－バイト操作

01－ワード操作

10－ロング・ワード操作

実効アドレス・フィールド――デスティネーション・ロケーションを指定する。次に示すとおり、可変アドレッシング・モードのみ可。

アドレッシング・モード	モード	レジスタ
Dn	000	reg. number : Dn
An*	001	reg. number : An
(An)	010	reg. number : An
(An)+	011	reg. number : An
−(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

アドレッシング・モード	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	−	−
(d_{16},PC)	−	−
(d_8,PC,Xn)	−	−
(bd,PC,Xn)	−	−
([bd,PC,Xn],od)	−	−
([bd,PC],Xn,od)	−	−

*ワードおよびロング・ワードのみ。

ADDX

Add Extended • 拡張加算

操作：　ソース＋デスティネーション＋X→デスティネーション

アセンブラ・シンタックス：　ADDX　Dy, Dx
ADDX −(Ay), −(Ax)

属性：　サイズ＝(バイト、ワード、ロング・ワード)

説明：　ソース・オペランドを拡張ビット(X)とともにデスティネーション・オペランドに加算し、結果をデスティネーション・ロケーションに格納します。オペランドは次の2とおりの方法によりアドレス指定できます。

1. データ・レジスタとデータ・レジスタ：オペランドは命令で指定するデータ・レジスタにあります。
2. メモリとメモリ：命令で指定するアドレス・レジスタを用いたプリデクリメント・アドレッシング・モードでオペランドを指定します。

操作サイズはバイト、ワード、ロング・ワードが指定できます。

コンディション・コード：

X	N	Z	V	C
*	*	*	*	*

X－キャリ・ビットと同じ。
N－結果が負であればセット、それ以外のときはクリア。
Z－結果が0でなければクリア、それ以外のときは変化しない。
V－オーバフローが発生すればセット、それ以外のときはクリア。
C－キャリが発生すればセット、それ以外のときはクリア。
注：通常、コンディション・コードのZビットは、演算を実行する前にプログラムでセットされるため、このビットを使用して多倍精度演算を終了したとき演算結果がゼロかどうかテストすることができます。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	レジスタRx			1	サイズ		0	0	R/M	レジスタRy		

命令フィールド：

レジスタRxフィールド――デスティネーションのレジスタを指定。
R/M＝0　データ・レジスタを指定。
R/M＝1　プリデクリメント・アドレッシング・モードで使用するアドレス・レジスタを指定。
サイズ・フィールド――操作サイズを指定。
00－バイト操作

01－ワード操作

10－ロング・ワード操作

R/Mフィールド――オペランドのアドレッシング・モードを指定。

0－データ・レジスタとデータ・レジスタの操作。

1－メモリとメモリの操作。

レジスタRyフィールド――ソース・レジスタを指定。

R/M＝0　データ・レジスタを指定。

R/M＝1　プリデクリメント・アドレス・モードを使用するアドレス・レジスタを指定。

AND

AND Logical・論理積

操作：　ソース∧デスティネーション→デスティネーション

アセンブラ・シンタックス：　AND ＜ea＞, Dn
AND Dn, ＜ea＞

属性：　サイズ＝(バイト、ワード、ロング・ワード)

説明：　ソース・オペランドとデスティネーション・オペランドの論理積をとり、結果をデスティネーション・ロケーションに格納します。操作サイズはバイト、ワード、ロング・ワードが指定できます。アドレス・レジスタの内容をオペランドとして使用することはできません。

コンディション・コード：

X	N	Z	V	C
—	*	*	0	0

X－影響を受けない。

N－結果の最上位ビットがセットされていればセット、それ以外のときはクリア。

Z－結果が0であればセット、それ以外のときはクリア。

V－常にクリア。

C－常にクリア。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	レジスタ			Opモード			実行アドレス					
										モード			レジスタ		

命令フィールド：

レジスタ・フィールド——8つのデータ・レジスタのいずれかを指定。

Opモード・フィールド：

バイト	ワード	ロング・ワード	操　作
000	001	010	(＜ea＞) ∧ (＜Dn＞) →Dn
100	101	110	(＜Dn＞) ∧ (＜ea＞) →ea

実効アドレス・フィールド——アドレッシング・モードを決定。
ソース・オペランドが指定された場合は、次に示すとおりデータ・アドレッシング・モードのみ可。

アドレッシング・モード	モード	レジスタ
Dn	000	reg. number : Dn
An	–	–
(An)	010	reg. number : An
(An)+	011	reg. number : An
–(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

アドレッシング・モード	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

デスティネーション・オペランドが指定された場合は、次に示すとおりメモリ可変アドレッシング・モードのみ可。

アドレッシング・モード	モード	レジスタ
Dn	–	–
An	–	–
(An)	010	reg. number : An
(An)+	011	reg. number : An
–(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

アドレッシング・モード	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	–	–
(d_{16},PC)	–	–
(d_8,PC,Xn)	–	–
(bd,PC,Xn)	–	–
([bd,PC,Xn],od)	–	–
([bd,PC],Xn,od)	–	–

注：1. デスティネーションがデータ・レジスタの場合、Dnがデスティネーションとなるモードを使用します。データ・レジスタに対しては、<ea>がデスティネーションとなるモードは無効です。
2. ソースがイミディエイト・データのときには、ほとんどのアセンブラがANDIを使用します。

ANDI

AND Immediate・イミディエイト論理積

操作：　イミディエイト・データ∧デスティネーション→デスティネーション

アセンブラ・シンタックス：　ANDI #<data>, <ea>

属性：　サイズ＝(バイト、ワード、ロング・ワード)

説明：　イミディエイト・データとデスティネーション・オペランドの論理積をとり、結果をデスティネーション・ロケーションに格納します。操作サイズは、バイト、ワード、ロング・ワードが指定できます。イミディエイト・データのサイズは操作サイズと同じです。

第3章　命令セット

コンディション・コード：

X	N	Z	V	C
—	*	*	0	0

X－影響を受けない。

N－結果の最上位ビットがセットされていればセット、それ以外のときはクリア。

Z－結果が0であればセット、それ以外のときはクリア。

V－常にクリア。

C－常にクリア。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	サイズ		実効アドレス モード			実効アドレス レジスタ		
ワード・データ（16ビット）								バイト・データ（8ビット）							
ロング・データ（32ビット）															

命令フィールド：

サイズ・フィールド――操作サイズを指定。

00－バイト操作

01－ワード操作

10－ロング・ワード操作

実効アドレス・フィールド――デスティネーションのオペランドを指定。次に示すとおり、データ可変アドレッシング・モードのみ可。

ｱﾄﾞﾚｯｼﾝｸﾞ･ﾓｰﾄﾞ	モード	レジスタ
Dn	000	reg. number : Dn
An	−	−
(An)	010	reg. number : An
(An)＋	011	reg. number : An
−(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

ｱﾄﾞﾚｯｼﾝｸﾞ･ﾓｰﾄﾞ	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	−	−
(d_{16},PC)	−	−
(d_8,PC,Xn)	−	−
(bd,PC,Xn)	−	−
([bd,PC,Xn],od)	−	−
([bd,PC],Xn,od)	−	−

イミディエイト・フィールド――（命令直後のデータ）

サイズ＝00　データはイミディエイト・ワードの下位バイト

サイズ＝01　データはイミディエイト・ワード全体

サイズ＝10　データは次の2つのイミディエイト・ワード

ANDI to CCR

AND Immediate to Condition Codes・コンディション・コードとのイミディエイト論理積

操作：　ソース∧CCR→CCR

アセンブラ・シンタックス：　ANDI #<data>, CCR

属性：　サイズ＝(バイト)

説明：　イミディエイト・オペランドとコンディション・コードの論理積をとり、結果をステータス・レジスタの下位のバイトに格納します。

コンディション・コード：

X	N	Z	V	C
*	*	*	*	*

X－イミディエイト・オペランドのビット4が0であればクリア、それ以外のときは変化しない。
N－イミディエイト・オペランドのビット3が0であればクリア、それ以外のときは変化しない。
Z－イミディエイト・オペランドのビット2が0であればクリア、それ以外のときは変化しない。
V－イミディエイト・オペランドのビット1が0であればクリア、それ以外のときは変化しない。
C－イミディエイト・オペランドのビット0が0であればクリア、それ以外のときは変化しない。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	0	0	1	1	1	1	0	0
0	0	0	0	0	0	0	0	バイト・データ（8ビット）							

ANDI to SR

AND Immediate to the Status Register(Privileged Instruction)・ステータス・レジスタとのイミディエイト論理積(特権命令)

操作：　スーパバイザ状態ではソース∧SR→SR
それ以外はTRAP

アセンブラ・シンタックス：　ANDI #<data>, SR

属性：　サイズ＝(ワード)

説明：　イミディエイト・オペランドとステータス・レジスタの内容の論理積をとり、結果をステータス・レジスタに格納します。ステータス・レジスタに実装されている全ビットが影響を受けます。

コンディション・コード：

X	N	Z	V	C
*	*	*	*	*

X－イミディエイト・オペランドのビット4が0であればクリア、それ以外のときは変化しない。
N－イミディエイト・オペランドのビット3が0であればクリア、それ以外のときは変化しない。
Z－イミディエイト・オペランドのビット2が0であればクリア、それ以外のときは変化しない。
V－イミディエイト・オペランドのビット1が0であればクリア、それ以外のときは変化しない。
C－イミディエイト・オペランドのビット0が0であればクリア、それ以外のときは変化しない。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	0	1	1	1	1	1	0	0
ワード・データ(16ビット)															

ASL、ASR

Arithmetic Shift・算術シフト

操作：　デスティネーション　Shifted by <カウント>→デスティネーション

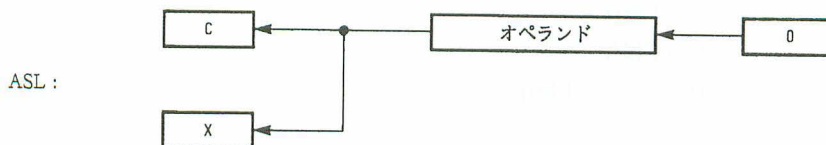
アセンブラ・シンタックス：　ASd Dx, Dy
ASd #<data>, Dy
ASd <ea>
dはシフト方向で、L（左）またはR（右）

属性：　サイズ＝(バイト、ワード、ロング・ワード)

説明：　オペランドのビットを指定方向（LまたはR）に算術シフトします。キャリ・ビットは、オペランドから最後に送り出されたビットを受け取ります。レジスタをシフトする場合のシフト回数は、次の2とおりの方法で指定できます。

1. イミディエイト――シフト回数は命令で指定する(シフト範囲、1～8)。
2. レジスタ――シフト回数は命令で指定するデータ・レジスタにある。モジュロ64。

操作サイズはバイト、ワード、ロング・ワードが指定できます。ただし、メモリ内のオペランドは1ビットしかシフトできず、操作サイズもワードに限定されます。

ASLではオペランドは左にシフトされ、シフトされた位置数がシフト回数になります。最上位から送り出されたビットはキャリと拡張の両方のビットに入り、最下位ビットには0が入ります。シフト中に符号が変化した場合は、オーバフロー・ビットがセットされます。

ASRではオペランドは右にシフトされ、シフトされた位置数がシフト回数になります。最下位から送り出されたビットはキャリと拡張の両方のビットに入り、符号ビットは再び最上位ビットに入ります。

コンディション・コード：

X	N	Z	V	C
*	*	*	*	*

X－オペランドから最後に送り出されたビットに従ってセット、シフト回数が0のときは変化しない。
N－結果の最上位ビットがセットされればセット、それ以外のときはクリア。
Z－結果が0であればセット、それ以外のときはクリア。
V－シフト操作中に最上位ビットが1回でも変化すればセット、それ以外のときはクリア。
C－オペランドから最後に送り出されたビットに従ってセット、シフト回数が0のときはクリア。

命令フォーマット(レジスタ・シフト)：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	カウント/レジスタ			dr	サイズ		i/r	0	0	レジスタ		

命令フィールド(レジスタ・シフト)：

カウント/レジスタ・フィールド――シフト回数またはシフト回数が入っているレジスタを指定。
　i/r＝0　このフィールドでシフト回数を指定、1～7は1～7、0は8を表わす。
　i/r＝1　このフィールドはシフト回数(モジュロ64)をもつデータ・レジスタを指定する。
dr フィールド――シフトの方向を指定。
　0－右シフト
　1－左シフト
サイズ・フィールド――操作サイズを指定。
　00－バイト操作
　01－ワード操作
　10－ロング・ワード操作
i/r フィールド
　i/r＝0　シフト回数をイミディエイト値で指定
　i/r＝1　シフト回数をレジスタで指定
レジスタ・フィールド――シフトするデータ・レジスタを指定。

命令フォーマット(メモリ・シフト)：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
										実効アドレス					
1	1	1	0	0	0	0	dr	1	1	モード			レジスタ		

命令フィールド(メモリ・シフト)：

dr フィールド――シフトの方向を指定。
　0－右シフト
　1－左シフト

実効アドレス・フィールド――シフトするオペランドを指定。次に示すとおりメモリ可変アドレッシング・モードのみ可。

アドレッシング・モード	モード	レジスタ
Dn	–	–
An	–	–
(An)	010	reg. number : An
(An)+	011	reg. number : An
–(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

アドレッシング・モード	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#<data>	–	–
(d_{16},PC)	–	–
(d_8,PC,Xn)	–	–
(bd,PC,Xn)	–	–
([bd,PC,Xn],od)	–	–
([bd,PC],Xn,od)	–	–

Bcc

Branch Conditionally・条件分岐

操作：　If（条件が真）then PC ＋d→PC

アセンブラ・シンタックス：　Bcc <label>

属性：　サイズ＝(バイト、ワード、ロング・ワード)

説明：　指定された条件が真の場合、プログラムの実行は（PC）＋ディスプレースメントに分岐します。PCはBcc命令の命令ワード＋2のアドレスを保持しています。ディスプレースメントは2の補数の整数で、現在のPCからデスティネーションPCまでの相対距離をバイトで表わします。命令ワードの8ビット・ディスプレースメント・フィールドが0の場合は、16ビットのディスプレースメント（命令直後のワード）が使用されます。命令ワードの8ビット・ディスプレースメント・フィールドがすべて1（＄FF）の場合は、32ビットのディスプレースメント（命令直後のロング・ワード）が使用されます。コンディション・コードccは、次の条件の1つを指定します。

CC	キャリ・クリア	0100	$\overline{C}$	LS	ローか同じ	0011	$C+Z$
CS	キャリ・セット	0101	C	LT	より小さい	1101	$N\cdot\overline{V}+\overline{N}\cdot V$
EQ	等しい	0111	Z	MI	マイナス	1011	N
GE	大きいか等しい	1100	$N\cdot V+\overline{N}\cdot\overline{V}$	NE	等しくない	0110	$\overline{Z}$
GT	より大き	1110	$N\cdot V\cdot\overline{Z}+\overline{N}\cdot\overline{V}\cdot\overline{Z}$	PL	プラス	1010	$\overline{N}$
HI	ハイ	0010	$\overline{C}\cdot\overline{Z}$	VC	オーバフロー・クリア	1000	$\overline{V}$
LE	小さいか等しい	1111	$Z+N\cdot\overline{V}+\overline{N}\cdot V$	VS	オーバフロー・セット	1001	V

コンディション・コード：　影響を受けない。

命令フォーマット：

<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>0</td><td colspan="4">コンディション</td><td colspan="8">8ビット・ディスプレースメント</td></tr>
<tr><td colspan="16">8ビット・ディスプレースメント＝＄00の場合、16ビット・ディスプレースメント</td></tr>
<tr><td colspan="16">8ビット・ディスプレースメント＝＄FFの場合、32ビット・ディスプレースメント</td></tr>
</table>

命令フィールド：

コンディション・フィールド――上記の14のコードのいずれかを指定する。

8ビット・ディスプレースメント・フィールド――分岐命令と条件が満たされた場合に次に実行すべき命令との相対距離（バイト単位）を示す2の補数の整数。

16ビット・ディスプレースメント・フィールド――8ビット・ディスプレースメント・フィールドに＄00が入れられているときに、ディスプレースメントに使用する。

32ビット・ディスプレースメント・フィールド――8ビット・ディスプレースメント・フィールドに＄FFが入れられているとき、ディスプレースメントに使用する。

注：すぐ後の命令に分岐するような短い分岐を行なうことはできない。この場合、8ビットのディスプレースメントのオフセットが0となり、ワード分岐命令の定義となるためです。

BCHG

Test a Bit and Change・ビット・テストと変更

操作：　デスティネーションの～（<ビット番号>）→Z；
　　　　デスティネーションの～（<ビット番号>）→デスティネーションの<ビット番号>

アセンブラ・シンタックス：　BCHG Dn, < ea >
　　　　BCHG #< data >, < ea >

属性：　サイズ=(バイト、ロング・ワード)

説明：　デスティネーション・オペランドの任意の1ビットをテストし、その状態をコンディション・コードZに反映します。テストの後、デスティネーションのそのビットの状態を反転します。デスティネーションがデータ・レジスタの場合、ビット番号はモジュロ32ビット番号で、32ビットのうちの任意のビットを指定することができます。デスティネーションがメモリ・ロケーションの場合、操作はバイト操作であり、ビット番号はモジュロ8です。いずれの場合にも、ビット0が最下位ビットです。この操作のビット番号は次の2とおりの方法で指定できます。

1. イミディエイト：ビット番号を命令の第2ワードで指定します。
2. レジスタ：ビット番号は命令で指定されるデータ・レジスタにあります。

コンディション・コード：

X	N	Z	V	C
—	—	*	—	—

X－影響を受けない。
N－影響を受けない。
Z－テストされたビットが0であればセット、そうでなければクリア。
V－影響を受けない。
C－影響を受けない。

命令フォーマット(ビット番号ダイナミック、レジスタで指定)：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	レジスタ			1	0	1	実効アドレス					
										モード			レジスタ		

命令フィールド(ビット番号ダイナミック)：
　　レジスタ・フィールド――ビット番号をもつデータ・レジスタ指定。

実効アドレス・フィールド――デスティネーション・ロケーションを指定。次に示すとおり、データ可変アドレッシング・モードのみ可。

アドレッシング・モード	モード	レジスタ
Dn*	000	reg. number : Dn
An	–	–
(An)	010	reg. number : An
(An)+	011	reg. number : An
–(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

アドレッシング・モード	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#<data>	–	–
(d_{16},PC)	–	–
(d_8,PC,Xn)	–	–
(bd,PC,Xn)	–	–
([bd,PC,Xn],od)	–	–
([bd,PC],Xn,od)	–	–

*ロング・ワードのみ、その他はすべてバイトのみ。

命令フォーマット(ビット番号スタティック、イミディエイト・データとして指定):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	0	1	実効アドレス モード			実効アドレス レジスタ		
0	0	0	0	0	0	0	0	ビット番号							

命令フィールド(ビット番号スタティック):

実効アドレス・フィールド――デスティネーション・ロケーションを指定。次に示すとおり、データ可変アドレッシング・モードのみ可。

アドレッシング・モード	モード	レジスタ
Dn*	000	reg. number : Dn
An	–	–
(An)	010	reg. number : An
(An)+	011	reg. number : An
–(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

アドレッシング・モード	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#<data>	–	–
(d_{16},PC)	–	–
(d_8,PC,Xn)	–	–
(bd,PC,Xn)	–	–
([bd,PC,Xn],od)	–	–
([bd,PC],Xn,od)	–	–

*ロング・ワードのみ、その他はすべてバイトのみ。

ビット番号フィールド――ビット番号を指定。

BCLR

Test a Bit and Clear・ビット・テストとクリア

操作：　～（デスティネーションの<ビット番号>）→Z；
0→デスティネーションの<ビット番号>

アセンブラ・シンタックス：　BLCR　dn, <ea>
BLCR　#<data>, <ea>

属性：　サイズ＝(バイト、ロング・ワード)

説明：　デスティネーションのオペランドの1ビットをテストし、そのビットの状態をコンディション・コードZに反映します。テストの後、デスティネーションのそのビットをクリアします。デスティネーションがデータ・レジスタの場合、モジュロ32のビット番号で32ビットの任意の1ビットを指定することができます。メモリ・ロケーションがデスティネーションの場合、操作はバイト操作となり、ビット番号はモジュロ8になります。いずれの場合にも、ビット0が最下位ビットです。この操作のビット番号は次の2とおりの方法で指定できます。

1. イミディエイト：ビット番号を命令の第2ワードで指定します。
2. レジスタ：ビット番号は命令で指定するデータ・レジスタにあります。

コンディション・コード：

X	N	Z	V	C
—	—	*	—	—

X－影響を受けない。
N－影響を受けない。
Z－テストされたビットが0であればセット、それ以外のときはクリア。
V－影響を受けない。
C－影響を受けない。

命令フォーマット(ビット番号ダイナミック、レジスタで指定)：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	レジスタ			1	1	0	実効アドレス					
										モード			レジスタ		

命令フィールド(ビット番号ダイナミック)：

レジスタ・フィールド――ビット番号をもつデータ・レジスタを指定。

実効アドレス・フィールド――デスティネーション・ロケーションを指定。次に示すとおり、データ可変アドレッシング・モードのみ可。

アドレッシング・モード	モード	レジスタ
Dn*	000	reg. number : Dn
An	–	–
(An)	010	reg. number : An
(An)+	011	reg. number : An
–(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

アドレッシング・モード	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	–	–
(d_{16},PC)	–	–
(d_8,PC,Xn)	–	–
(bd,PC,Xn)	–	–
([bd,PC,Xn],od)	–	–
([bd,PC],Xn,od)	–	–

*ロング・ワードのみ、その他はすべてバイトのみ。

命令フォーマット(ビット番号スタティック、イミディエイト・データとして指定):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	1	0	実効アドレス モード			実効アドレス レジスタ		
0	0	0	0	0	0	0	0	ビット番号							

命令フィールド(ビット番号スタティック):

実効アドレス・フィールド――デスティネーション・ロケーションを指定。次に示すとおり可変アドレッシング・モードのみ可。

アドレッシング・モード	モード	レジスタ
Dn*	000	reg. number : Dn
An	–	–
(An)	010	reg. number : An
(An)+	011	reg. number : An
–(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

アドレッシング・モード	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	–	–
(d_{16},PC)	–	–
(d_8,PC,Xn)	–	–
(bd,PC,Xn)	–	–
([bd,PC,Xn],od)	–	–
([bd,PC],Xn,od)	–	–

*ロング・ワードのみ、その他はすべてバイトのみ。

ビット番号フィールド――ビット番号を指定。

BFCHG

Test Bit Field and Change・ビット・フィールドのテストおよび変更

操作：　～（デスティネーションの<ビット・フィールド>）→デスティネーションの<ビット・フィールド>

アセンブラ・シンタックス：　BFCHG　<ea>｛オフセット：幅｝

属性：　サイズなし

説明：　指定された実効アドレスにあるビット・フィールドの値に従ってコンディション・コードをセットし、ついでそのフィールドを補数化します。
フィールド・オフセットおよびフィールド幅によってフィールドを選択します。フィールド・オフセットは、そのフィールドの開始ビットを指定します。フィールド幅でそのフィールドのビット数を決定します。

コンディション・コード：

X	N	Z	V	C
—	*	*	0	0

X－影響を受けない。
N－フィールドの最上位ビットがセットされていればセット、それ以外のときはクリア。
Z－フィールドの全ビットが0であればセット、それ以外のときはクリア。
V－常にクリア。
C－常にクリア。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	0	1	0	1	1	実効アドレス モード			実効アドレス レジスタ		
0	0	0	0	Do	オフセット					Dw	幅				

命令フィールド：

実効アドレス・フィールド――そのビット・フィールドのベース・ロケーションを指定。次に示すようにデータ・レジスタ直接または制御可変アドレッシング・モードのみ可。

アドレッシング・モード	モード	レジスタ
Dn	000	reg. number : Dn
An	–	–
(An)	010	reg. number : An
(An)+	–	–
–(An)	–	–
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

アドレッシング・モード	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	–	–
(d_{16},PC)	–	–
(d_8,PC,Xn)	–	–
(bd,PC,Xn)	–	–
([bd,PC,Xn],od)	–	–
([bd,PC],Xn,od)	–	–

Doフィールド――フィールド・オフセットの指定方法を決定する。

0－ビット・フィールド・オフセットはオフセット・フィールドにある。

1－拡張ワードのビット［8：6］がオフセットを含むデータ・レジスタを指定。ビット［10：9］は0。

オフセット・フィールド――Doに従って、フィールド・オフセットを指定。

Do＝0　オフセット・フィールドはイミディエイトオペランド。オペランド値の範囲は0～31。

Do＝1　オフセット・フィールドはオフセットをもつデータ・レジスタを指定。値の範囲は-2^{31}～$+2^{31}-1$。

Dwフィールド――フィールド幅の指定方法を決定する。

0－ビット・フィールド幅は幅フィールドにある。

1－拡張ワードのビット［2：0］が幅の値をもつデータ・レジスタを指定。ビット［3：4］は0。

幅フィールド――Dwに従って、フィールド幅を指定。

Dw＝0　幅フィールドは、イミディエイト・オペランド。1～31の範囲のオペランド値が1～31のフィールド幅を指定。0の値は32の幅を指定。

Dw＝1　幅フィールドは、幅の値をもつデータ・レジスタを指定。この値はモジュロ32であり、1～31の値は1～31のフィールド幅を指定し、0の値は32の幅を指定。

BFCLR

Test Bit Field and Clear・ビット・フィールドのテストとクリア

操作：　0→デスティネーションの<ビット・フィールド>

アセンブラ・シンタックス：　BFCLR <ea> {オフセット：幅}

属性：　サイズなし

説明：　指定された実効アドレスにあるビット・フィールドの値に従ってコンディション・コードをセットし、ついでそのフィールドをクリアします。

フィールド・オフセットおよびフィールド幅によってフィールドを選択します。フィールド・オフセットは、そのフィールドの開始ビットを指定します。フィールド幅でそのフィールドのビット数を決定します。

コンディション・コード：

X	N	Z	V	C
—	*	*	0	0

X－影響を受けない。

N－フィールドの最上位ビットがセットされていればセット、それ以外のときはクリア。

Z－フィールドの全ビットが0であればセット、それ以外のときはクリア。

V－常にクリア。

C－常にクリア。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	1	0	0	1	1	実効アドレス モード			レジスタ		
0	0	0	0	Do	オフセット					Dw	幅				

命令フィールド：

実効アドレス・フィールド――そのビット・フィールドのベース・ロケーションを指定。次に示すようにデータ・レジスタ直接または制御可変アドレッシング・モードのみ可。

アドレッシング・モード	モード	レジスタ
Dn	000	reg. number : Dn
An	–	–
(An)	010	reg. number : An
(An)＋	–	–
–(An)	–	–
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

アドレッシング・モード	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	–	–
(d_{16},PC)	–	–
(d_8,PC,Xn)	–	–
(bd,PC,Xn)	–	–
([bd,PC,Xn],od)	–	–
([bd,PC],Xn,od)	–	–

Doフィールド――フィールド・オフセットの指定方法を決定する。

0－ビット・フィールド・オフセットはオフセット・フィールドにある。

1－拡張ワードのビット［8：6］がオフセットを含むデータ・レジスタを指定。ビット［10：9］は0。

オフセット・フィールド――Doに従って、フィールド・オフセットを指定。

Do＝0　オフセット・フィールドはイミディエイト・オペランド。オペランド値の範囲は0～31。

Do＝1　オフセット・フィールドはオフセットをもつデータ・レジスタを指定。値の範囲は-2^{31}～$+2^{31}-1$。

Dwフィールド――フィールド幅の指定方法を決定する。

0－ビット・フィールド幅は幅フィールドにある。

1－拡張ワードのビット［2：0］が幅の値をもつデータ・レジスタを指定。ビット［3：4］は0。

幅フィールド――Dwに従って、フィールド幅を指定。

Dw＝0　幅フィールドは、イミディエイト・オペランド。1～31の範囲のオペランド値が1～31のフィールド幅を指定。0の値は32の幅を指定。

Dw＝1　幅フィールドは、幅の値をもつデータ・レジスタを指定。この値はモジュロ32であり、1～31の値は1～31のフィールド幅を指定し、0の値は32の幅を指定。

BFEXTS

Extract Bit Field Signed・符号付きビット・フィールド抽出

操作：　ソースの<ビット・フィールド>→Dn

アセンブラ・シンタックス：　BFEXTS <ea> {オフセット：幅}, Dn

属性：　サイズなし

説明：　指定された実効アドレスのロケーションからビット・フィールドを抽出し、32ビットに符号拡張した後、その結果をデスティネーション・データ・レジスタにロードします。
フィールド・オフセットおよびフィールド幅によってフィールドを選択します。フィールド・オフセットは、そのフィールドの開始ビットを指定します。フィールド幅でそのフィールドのビット数を決定します。

コンディション・コード：

X	N	Z	V	C
—	*	*	0	0

X－影響を受けない。

N－フィールドの最上位ビットがセットされていればセット、それ以外のときはクリア。

Z－フィールドの全ビットが0であればセット、それ以外のときはクリア。

V－常にクリア。

C－常にクリア。

命令フォーマット：

<table>
<tr><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td rowspan="2">1</td><td rowspan="2">1</td><td rowspan="2">1</td><td rowspan="2">0</td><td rowspan="2">1</td><td rowspan="2">0</td><td rowspan="2">1</td><td rowspan="2">1</td><td rowspan="2">1</td><td rowspan="2">1</td><td colspan="6">実効アドレス</td></tr>
<tr><td colspan="3">モード</td><td colspan="3">レジスタ</td></tr>
<tr><td>0</td><td colspan="3">レジスタ</td><td>Do</td><td colspan="5">オフセット</td><td>Dw</td><td colspan="5">幅</td></tr>
</table>

命令フィールド：

実効アドレス・フィールド――そのビット・フィールドのベース・ロケーションを指定。次に示すようにデータ・レジスタ直接または制御アドレッシング・モードのみ可。

ｱﾄﾞﾚｯｼﾝｸﾞ･ﾓｰﾄﾞ	モード	レジスタ
Dn	000	reg. number : Dn
An	–	–
(An)	010	reg. number : An
(An)+	–	–
–(An)	–	–
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

ｱﾄﾞﾚｯｼﾝｸﾞ･ﾓｰﾄﾞ	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	–	–
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

レジスタ・フィールド――デスティネーション・レジスタを指定。

Do フィールド――フィールド・オフセットの指定方法を決定する。

0－ビット・フィールド・オフセットはオフセット・フィールドにある。

1－拡張ワードのビット［8：6］がオフセットを含むデータ・レジスタを指定。ビット［10：9］は0。

オフセット・フィールド――Do に従って、フィールド・オフセットを指定。

Do＝0　オフセット・フィールドはイミディエイト・オペランド。オペランド値の範囲は0～31。

Do＝1　オフセット・フィールドはオフセットをもつデータ・レジスタを指定。値の範囲は-2^{31}～$+2^{31}-1$。

Dw フィールド――フィールド幅の指定方法を決定する。

0－ビット・フィールド幅は幅フィールドにある。

1－拡張ワードのビット［2：0］が幅の値をもつデータ・レジスタを指定。ビット［4：3］は0。

幅フィールド――Dw に従って、フィールド幅を指定。

Dw＝0　幅フィールドは、イミディエイト・オペランド。1～31の範囲のオペランド値が1～31のフィールド幅を指定。0の値は32の幅を指定。

Dw＝1　幅フィールドは、幅の値をもつデータ・レジスタを指定。この値はモジュロ32であり、1～31の値は1～31のフィールド幅を指定し、0の値は32の幅を指定。

BFEXTU

Extract Bit Field Unsigned • 符号なしビット・フィールド抽出

操作：　ソースの＜ビット・フィールド＞→Dn

アセンブラ・シンタックス：　BFEXTU＜ea＞｛オフセット：幅｝, Dn

属性：　サイズなし

説明：　指定された実効アドレス・ロケーションからビット・フィールドを抽出し、32ビットにゼロ拡張した後、その結果をデスティネーション・データ・レジスタにロードします。

フィールド・オフセットおよびフィールド幅によってフィールドを選択します。フィールド・オフセットは、そのフィールドの開始ビットを指定します。フィールド幅でそのフィールドのビット数を決定します。

コンディション・コード：

X	N	Z	V	C
—	*	*	0	0

X－影響を受けない。

N－フィールドの最上位ビットがセットされていればセット、それ以外のときはクリア。

Z－フィールドの全ビットが0であればセット、それ以外のときはクリア。

V－常にクリア。

C－常にクリア。

命令フォーマット：

<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td rowspan="2">1</td><td rowspan="2">1</td><td rowspan="2">1</td><td rowspan="2">0</td><td rowspan="2">1</td><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">1</td><td rowspan="2">1</td><td rowspan="2">1</td><td colspan="6">実効アドレス</td></tr>
<tr><td colspan="3">モード</td><td colspan="3">レジスタ</td></tr>
<tr><td>0</td><td colspan="3">レジスタ</td><td>Do</td><td colspan="5">オフセット</td><td>Dw</td><td colspan="5">幅</td></tr>
</table>

命令フィールド：

実効アドレス・フィールド――そのビット・フィールドのベース・ロケーションを指定。次に示すようにデータ・レジスタ直接または制御アドレッシング・モードのみ可。

アドレッシング・モード	モード	レジスタ
Dn	000	reg. number : Dn
An	–	–
(An)	010	reg. number : An
(An)+	–	–
–(An)	–	–
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

アドレッシング・モード	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#<data>	–	–
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

レジスタ・フィールド――デスティネーション・レジスタを指定。

Do フィールド――フィールド・オフセットの指定方法を決定する。

0－ビット・フィールド・オフセットはオフセット・フィールドにある。

1－拡張ワードのビット［8：6］がオフセットを含むデータ・レジスタを指定。ビット［10：9］は0。

オフセット・フィールド――Do に従って、フィールド・オフセットを指定。

Do＝0　オフセット・フィールドはイミディエイト・オペランド。オペランド値の範囲は0～31。

Do＝1　オフセット・フィールドはオフセットをもつデータ・レジスタを指定。値の範囲は-2^{31}～$2^{31}-1$。

Dw フィールド――フィールド幅の指定方法を決定する。

0－ビット・フィールド幅は幅フィールドにある。

1－拡張ワードのビット［2：0］が幅の値をもつデータ・レジスタを指定。ビット［4：3］は0。

幅フィールド――Dw に従って、フィールド幅を指定。

Dw＝0　幅フィールドは、イミディエイト・オペランド。1～31の範囲のオペランド値が1～31のフィールド幅を指定。0の値は32の幅を指定。

Dw＝1　幅フィールドは、幅の値をもつデータ・レジスタを指定。この値はモジュロ32であり、1～31の値は1～31のフィールド幅を指定し、0の値は32の幅を指定。

BFFFO

Find First One in Bit Field・ビット・フィールド内の最初の１検出

操作：　ソース・ビット走査の<ビット・オフセット>→Dn

アセンブラ・シンタックス：　BFFFO < ea > {オフセット：幅},Dn

属性：　サイズなし

説明：　ソース・オペランドの中で、1にセットされている最上位ビットの位置を求めます。そのビットのビット・オフセット(命令でのビット・オフセット＋最初のセット・ビットのオフセット)がDnに入れられます。そのビット・フィールドでどのビットも1にセットされていない場合は、Dnの値がフィールド・オフセット＋フィールド幅になります。ビット・フィールド値に従ってコンディション・コードがセットされます。

フィールド・オフセットおよびフィールド幅によってフィールドを選択します。フィールド・オフセットは、そのフィールドの開始ビットを指定します。フィールド幅でそのフィールドのビット数を決定します。

コンディション・コード：

X	N	Z	V	C
—	*	*	0	0

X－影響を受けない。

N－フィールドの最上位ビットがセットされていればセット、それ以外のときはクリア。

Z－フィールドの全ビットが0であればセット、それ以外のときはクリア。

V－常にクリア。

C－常にクリア。

命令フォーマット：

<table>
<tr><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td rowspan="2">1</td><td rowspan="2">1</td><td rowspan="2">1</td><td rowspan="2">0</td><td rowspan="2">1</td><td rowspan="2">1</td><td rowspan="2">0</td><td rowspan="2">1</td><td rowspan="2">1</td><td rowspan="2">1</td><td colspan="6">実効アドレス</td></tr>
<tr><td colspan="3">モード</td><td colspan="3">レジスタ</td></tr>
<tr><td>0</td><td colspan="3">レジスタ</td><td>Do</td><td colspan="5">オフセット</td><td>Dw</td><td colspan="5">幅</td></tr>
</table>

命令フィールド：

実効アドレス・フィールド──そのビット・フィールドのベース・ロケーションを指定。次に示すようにデータ・レジスタ直接または制御アドレッシング・モードのみ可。

ｱﾄﾞﾚｯｼﾝｸﾞ･ﾓｰﾄﾞ	モード	レジスタ
Dn	000	reg. number : Dn
An	−	−
(An)	010	reg. number : An
(An)+	−	−
−(An)	−	−
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

ｱﾄﾞﾚｯｼﾝｸﾞ･ﾓｰﾄﾞ	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#<data>	−	−
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

レジスタ・フィールド──デスティネーションのデータ・レジスタ・オペランドを指定。

Do フィールド──フィールド・オフセットの指定方法を決定する。

0－ビット・フィールド・オフセットはオフセット・フィールドにある。

1－拡張ワードのビット［8：6］がオフセットを含むデータ・レジスタを指定。ビット［10：9］は0。

オフセット・フィールド──Doに従って、フィールド・オフセットを指定。

Do＝0　オフセット・フィールドはイミディエイト・オペランド。オペランド値の範囲は0～31。

Do＝1　オフセット・フィールドはオフセットをもつデータ・レジスタを指定。値の範囲は-2^{31}～$2^{31}-1$。

Dw フィールド──フィールド幅の指定方法を決定する。

0－ビット・フィールド幅は幅フィールドにある。

1－拡張ワードのビット［2：0］が幅の値をもつデータ・レジスタを指定。ビット［4：3］は0。

幅フィールド──Dwに従って、フィールド幅を指定。

Dw＝0　幅フィールドは、イミディエイト・オペランド。1～31の範囲のオペランド値が1～31のフィールド幅を指定。0の値は32の幅を指定。

Dw＝1　幅フィールドは、幅の値をもつデータ・レジスタを指定。この値はモジュロ32であり、1～31の値は1～31のフィールド幅を指定し、0の値は32の幅を指定。

BFINS

Insert Bit Field・ビット・フィールド挿入

操作：　Dn→デスティネーションの＜ビット・フィールド＞

アセンブラ・シンタックス：　BFINS Dn, ＜ea＞ {オフセット：幅}

属性：　サイズなし

説明：　指定されたデータ・レジスタの下位ビットからビット・フィールドを取り出して、指定された実効アドレス・ロケーションのビット・フィールドに転送します。コンディション・コードは、挿入された値に従ってセットされます。

フィールド・オフセットおよびフィールド幅によってフィールドを選択します。フィールド・オフセットは、そのフィールドの開始ビットを指定します。フィールド幅でそのフィールドのビット数を決定します。

コンディション・コード：

X	N	Z	V	C
—	*	*	0	0

X－影響を受けない。

N－フィールドの最上位ビットがセットされていればセット、それ以外のときはクリア。

Z－フィールドの全ビットが0であればセット、それ以外のときはクリア。

V－常にクリア。

C－常にクリア。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	1	1	1	1	1	実効アドレス					
										モード			レジスタ		
0	レジスタ			Do	オフセット					Dw	幅				

命令フィールド：

実効アドレス・フィールド——そのビット・フィールドのベース・ロケーションを指定。次に示すようにデータ・レジスタ直接または制御可変アドレッシング・モードのみ可。

ｱﾄﾞﾚｯｼﾝｸﾞ・ﾓｰﾄﾞ	モード	レジスタ
Dn	000	reg. number : Dn
An	–	–
(An)	010	reg. number : An
(An)+	–	–
–(An)	–	–
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

ｱﾄﾞﾚｯｼﾝｸﾞ・ﾓｰﾄﾞ	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	–	–
(d_{16},PC)	–	–
(d_8,PC,Xn)	–	–
(bd,PC,Xn)	–	–
([bd,PC,Xn],od)	–	–
([bd,PC],Xn,od)	–	–

レジスタ・フィールド——ソース・データ・レジスタ・オペランドを指定。

Doフィールド——フィールド・オフセットの指定方法を決定する。

0－ビット・フィールド・オフセットはオフセット・フィールドにある。

1－拡張ワードのビット［8：6］がオフセットを含むデータ・レジスタを指定。ビット［10：9］は0。

オフセット・フィールド——Doに従って、フィールド・オフセットを指定。

Do＝0　オフセット・フィールドはイミディエイト・オペランド。オペランド値の範囲は0～31。

Do＝1　オフセット・フィールドはオフセットをもつデータ・レジスタを指定。値の範囲は-2^{31}～$2^{31}-1$。

Dwフィールド——フィールド幅の指定方法を決定する。

0－ビット・フィールド幅は幅フィールドにある。

1－拡張ワードのビット［2：0］が幅の値をもつデータ・レジスタを指定。ビット［4：3］は0。

幅フィールド——Dwに従って、フィールド幅を指定。

Dw＝0　幅フィールドは、イミディエイト・オペランド。1～31の範囲のオペランド値が1～31のフィールド幅を指定。0の値は32の幅を指定。

Dw＝1　幅フィールドは、幅の値をもつデータ・レジスタを指定。この値はモジュロ32であり、1～31の値は1～31のフィールド幅を指定し、0の値は32の幅を指定。

BFSET

Test Bit Field and Set・ビット・フィールド・セット

操作：　1s→デスティネーションの<ビット・フィールド>

アセンブラ・シンタックス：　BFSET ＜ea＞ {オフセット：幅}

属性：　サイズなし

説明：　指定された実効アドレスのロケーションにあるビット・フィールドの全ビットをセットします。コンディション・コードは、セットされる前の値に従ってセットされます。

フィールド・オフセットおよびフィールド幅によってフィールドを選択します。フィールド・オフセットは、そのフィールドの開始ビットを指定します。フィールド幅でそのフィールドのビット数を決定します。

コンディション・コード：

X	N	Z	V	C
—	*	*	0	0

X－影響を受けない。

N－フィールドの最上位ビットがセットされていればセット、それ以外のときはクリア。

Z－フィールドの全ビットが0であればセット、それ以外のときはクリア。

V－常にクリア。

C－常にクリア。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	1	1	0	1	1	実効アドレス モード			実効アドレス レジスタ		
0	0	0	0	Do	オフセット					Dw	幅				

命令フィールド：

実効アドレス・フィールド――ビット・フィールドのベース・ロケーションを指定。次に示すように データ・レジスタ直接または制御可変アドレッシング・モードのみ可。

アドレッシング・モード	モード	レジスタ
Dn	000	reg. number : Dn
An	–	–
(An)	010	reg. number : An
(An)＋	–	–
－(An)	–	–
$(d_{16}$,An)	101	reg. number : An
$(d_8$,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

アドレッシング・モード	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	–	–
$(d_{16}$,PC)	–	–
$(d_8$,PC,Xn)	–	–
(bd,PC,Xn)	–	–
([bd,PC,Xn],od)	–	–
([bd,PC],Xn,od)	–	–

Doフィールド――フィールド・オフセットの指定方法を決定する。

0－ビット・フィールド・オフセットはオフセット・フィールドにある。

1－拡張ワードのビット［8：6］がオフセットを含むデータ・レジスタを指定。ビット［10：9］は0。

オフセット・フィールド――Doに従って、フィールド・オフセットを指定。

Do＝0　オフセット・フィールドはイミディエイト・オペランド。オペランド値の範囲は0～31。

Do＝1　オフセット・フィールドはオフセットをもつデータ・レジスタを指定。値の範囲は-2^{31}～$2^{31}-1$。

Dwフィールド――フィールド幅の指定方法を決定する。

0－ビット・フィールド幅は幅フィールドにある。

1－拡張ワードのビット［2：0］が幅の値をもつデータ・レジスタを指定。ビット［4：3］は0。

幅フィールド――Dwに従って、フィールド幅を指定。

Dw＝0　幅フィールドは、イミディエイト・オペランド。1～31の範囲のオペランド値は1～31のフィールド幅を指定。0の値は32の幅を指定。

Dw＝1　幅フィールドは、幅の値をもつデータ・レジスタを指定。この値はモジュロ32であり、1～31の値は1～31のフィールド幅を指定し、0の値は32の幅を指定。

BFTST

Test Bit Field・ビット・フィールド・テスト

操作：　デスティネーションの<ビット・フィールド>

アセンブラ・シンタックス：　BFTST <ea> {オフセット：幅}

属性：　サイズなし

説明：　指定された実効アドレス・ロケーションにあるビット・フィールドの値に従ってコンディション・コードをセットします。

フィールド・オフセットおよびフィールド幅によってフィールドを選択します。フィールド・オフセットは、そのフィールドの開始ビットを指定します。フィールド幅でそのフィールドのビット数を決定します。

コンディション・コード：

X	N	Z	V	C
—	*	*	0	0

X－影響を受けない。

N－フィールドの最上位ビットがセットされていればセット、それ以外のときはクリア。

Z－フィールドの全ビットが0であればセット、それ以外のときはクリア。

V－常にクリア。

C－常にクリア。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	0	0	0	1	1	実効アドレス モード			レジスタ		
0	0	0	0	Do	オフセット					Dw	幅				

命令フィールド：

実効アドレス・フィールド——ビット・フィールドのベース・ロケーションを指定。次に示すようにデータ・レジスタ直接または制御アドレッシング・モードのみ可。

アドレッシング・モード	モード	レジスタ
Dn	000	reg. number : Dn
An	–	–
(An)	010	reg. number : An
(An)+	–	–
–(An)	–	–
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

アドレッシング・モード	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	–	–
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

Doフィールド——フィールド・オフセットの指定方法を決定する。

0－ビット・フィールド・オフセットはオフセット・フィールドにある。

1－拡張ワードのビット［8：6］がオフセットを含むデータ・レジスタを指定。ビット［10：9］は0。

オフセット・フィールド——Doに従って、フィールド・オフセットを指定。

Do＝0　オフセット・フィールドはイミディエイト・オペランド。オペランド値の範囲は0～31。

Do＝1　オフセット・フィールドはオフセットをもつデータ・レジスタを指定。値の範囲は-2^{31}～$2^{31}-1$。

Dwフィールド——フィールド幅の指定方法を決定する。

0－ビット・フィールド幅は幅フィールドにある。

1－拡張ワードのビット［2：0］が幅の値をもつデータ・レジスタを指定。ビット［4：3］は0。

幅フィールド——Dwに従って、フィールド幅を指定。

Dw＝0　幅フィールドは、イミディエイト・オペランド。1～31の範囲のオペランド値は1～31のフィールド幅を指定。0の値は32の幅を指定。

Dw＝1　幅フィールドは、幅の値をもつデータ・レジスタを指定。この値はモジュロ32であり、1～31の値は1～31のフィールド幅を指定し、0の値は32の幅を指定。

BKPT

Breakpoint • ブレークポイント

操作：　ブレークポイント・アクノリッジ・サイクルを実行する。
アクノリッジされた場合は、返されたオペレーション・ワードを実行し、そうでない場合は、不当命令としてトラップする。

アセンブラ・シンタックス：　BKPT #<data>

属性：　サイズなし

説明：　アドレス・バスのビット2～4にイミディエイト・データ(値0～7)、ビット0と1にゼロを出力して、ブレークポイント・アクノリッジ・バス・サイクルを実行します。

ブレークポイント・アクノリッジ・サイクルは、CPU空間にアクセスし、タイプ0のアドレッシングを行ない、さらに命令で指定されたブレークポイント番号をアドレス・ラインA2～A4に出力します。外部ハードウェアが$\overline{\text{DSACKxs}}$または$\overline{\text{STERM}}$でサイクルを終了した場合は、バス上のデータ(命令ワード)が命令パイプに挿入され、ブレークポイント命令の後で実行されます。ブレークポイント命令は転送するワードを必要としますので、最初のバス・サイクルが8ビット・ポートにアクセスする場合には、第2サイクルが必要です。外部ロジックが$\overline{\text{BERR}}$でブレークポイント・アクノリッジ・サイクルを終了した場合(つまり、命令ワードがないとき)、プロセッサは不当命令例外の処理を行ないます。ブレークポイント・アクノリッジ・サイクル動作の詳細については、「7. 4. 2　ブレークポイント・アクノリッジ・サイクル」を参照してください。

この命令は、デバッグ・モニタやリアルタイム・ハードウェア・エミュレータのブレークポイント機能をサポートします。正確な操作は、インプリメント方法によって異なります。通常、この命令はプログラム内の1命令を置き換え、その命令はブレークポイント・アクノリッジ・サイクルで返されます。

コンディション・コード：　影響を受けない。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	1	0	0	1	ベクタ		

命令フィールド：

ベクタ・フィールド――0～7の範囲のイミディエイト・データがあります。これはブレークポイント番号です。

BRA

Branch Always・無条件分岐

操作：　PC＋d →PC

アセンブラ・シンタックス：　BRA < label >

属性：　サイズ＝(バイト、ワード、ロング・ワード)

説明：　プログラムは(PC)＋ディスプレースメントに分岐します。PCはBRA命令の命令ワードのアドレス＋2を保持しています。このディスプレースメントは2の補数の整数で、現在のPCからデスティネーションのPCまでの相対距離をバイトで表わしたものです。命令ワードの8ビット・ディスプレースメントが0の場合は、16ビット・ディスプレースメント(命令直後のワード)が使用されます。命令ワードの8ビット・ディスプレースメントがすべて1($FF)の場合は、32ビット・ディスプレースメント(命令直後のロング・ワード)が使用されます。

コンディション・コード：　影響を受けない。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	0	8ビット・ディスプレースメント							
8ビット・ディスプレースメント＝$00の場合、16ビット・ディスプレースメント															
8ビット・ディスプレースメント＝$FFの場合、32ビット・ディスプレースメント															

命令フィールド：

8ビット・ディスプレースメント・フィールド――この分岐命令と次に実行する命令との間のバイト数を示す2の補数の整数。

16ビット・ディスプレースメント・フィールド――8ビット・ディスプレースメントが$00のときに、より大きなディスプレースメントを得るために使用。

32ビット・ディスプレースメント・フィールド――8ビット・ディスプレースメントが$FFのときに、より大きなディスプレースメントを得るために使用。

注：すぐ後の命令への分岐の場合、8ビット・ディスプレースメント・フィールドが$00（ゼロ・オフセット）となるため、自動的に16ビット・ディスプレースメントが使用されます。

BSET

Test a Bit and Set・ビット・テストとセット

操作：　～（デスティネーションの<ビット番号>）→Z；
1→デスティネーションの　<ビット番号>

アセンブラ・シンタックス：　BSET Dn, <ea>
BSET　#<data>, <ea>

属性：　サイズ＝(バイト、ロング・ワード)

説明：　デスティネーション・オペランドの1ビットをテストし、そのビット状態をコンディション・コードのZに反映します。テストの後、デスティネーション・オペランドのそのビットをセットします。データ・レジスタがデスティネーションの場合、モジュロ32のビット番号により、32ビットのうちの任意のビットを指定することができます。メモリ・ロケーションがデスティネーションの場合、操作はバイト操作になり、ビット番号はモジュロ8になります。いずれの場合にも、ビット0が最下位ビットです。この操作のビット番号は次の2とおりの方法で指定できます。

1. イミディエイト：ビット番号を命令の第2ワードで指定します。
2. レジスタ：ビット番号は命令で指定するデータ・レジスタにあります。

コンディション・コード：

X	N	Z	V	C
—	—	*	—	—

X－影響を受けない。
N－影響を受けない。
Z－テストされたビットが0であればセット、それ以外のときはクリア。
V－影響を受けない。
C－影響を受けない。

命令フォーマット（ビット番号ダイナミック、レジスタで指定）：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	レジスタ			1	1	1	実効アドレス					
										モード			レジスタ		

命令フィールド(ビット番号ダイナミック):

レジスタ・フィールド——ビット番号をもつデータ・レジスタを指定。

実効アドレス・フィールド——デスティネーション・ロケーションを指定。次に示すとおり、データ可変アドレッシング・モードのみ可。

アドレッシング・モード	モード	レジスタ
Dn*	000	reg. number : Dn
An	–	–
(An)	010	reg. number : An
(An)+	011	reg. number : An
–(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

アドレッシング・モード	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	–	–
(d_{16},PC)	–	–
(d_8,PC,Xn)	–	–
(bd,PC,Xn)	–	–
([bd,PC,Xn],od)	–	–
([bd,PC],Xn,od)	–	–

*ロング・ワードのみ、その他はすべてバイトのみ。

命令フォーマット(ビット番号スタティック、イミディエイト・データとして指定):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	1	1	実効アドレス モード			実効アドレス レジスタ		
0	0	0	0	0	0	0	ビット番号								

命令フィールド(ビット番号スタティック):

実効アドレス・フィールド——デスティネーション・ロケーションを指定、次に示すとおり、データ可変アドレッシング・モードのみ可。

アドレッシング・モード	モード	レジスタ
Dn*	000	reg. number : Dn
An	–	–
(An)	010	reg. number : An
(An)+	011	reg. number : An
–(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

アドレッシング・モード	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	–	–
(d_{16},PC)	–	–
(d_8,PC,Xn)	–	–
(bd,PC,Xn)	–	–
([bd,PC,Xn],od)	–	–
([bd,PC],Xn,od)	–	–

*ロング・ワードのみ、その他はすべてバイトのみ。

ビット番号フィールド——ビット番号を指定。

BSR

Branch to Subroutine・サブルーチン分岐

操作：　SP－4→SP；PC→（SP）；PC＋d→PC

アセンブラ・シンタックス：　BSR <label>

属性：　サイズ＝(バイト、ワード、ロング・ワード)

説明：　BSR 命令の直後の命令のロング・ワード・アドレスをシステム・スタックにプッシュします。PCには命令ワードのアドレス＋2が入ります。プログラムの実行は(PC)＋ディスプレースメントのロケーションから続行されます。ディスプレースメントは2の補数の整数で、現在のPCからデスティネーションPCまでの相対距離をバイトで表わしたものです。命令ワードの8ビット・ディスプレースメント・フィールドが0の場合は、16ビット・ディスプレースメント(命令直後のワード)が使用されます。命令ワードの8ビット・ディスプレースメントがすべて1($FF)の場合は、32ビット・ディスプレースメント(命令直後のロング・ワード)が使用されます。

コンディション・コード：　影響を受けない。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	0	8ビット・ディスプレースメント							
8ビット・ディスプレースメント＝$00の場合、16ビット・ディスプレースメント															
8ビット・ディスプレースメント＝$FFの場合、32ビット・ディスプレースメント															

命令フィールド：

8ビット・ディスプレースメント・フィールド――この分岐命令と次に実行する命令との間のバイト数を示す2の補数の整数。

16ビット・ディスプレースメント・フィールド――8ビット・ディスプレースメントが$00のときに、より大きなディスプレースメントを得るために使用。

32ビット・ディスプレースメント・フィールド――8ビット・ディスプレースメントが$FFのときに、より大きなディスプレースメントを得るために使用。

注：すぐ後の命令への分岐の場合、8ビット・ディスプレースメント・フィールドが$00(ゼロ・オフセット)となるため、自動的に16ビット・ディスプレースメントが使用されます。

BTST

Test a Bit **・ ビット・テスト**

操作：　～（デスティネーションの<ビット番号>）→Z；

アセンブラ・シンタックス：　BTST Dn, <ea>

BTST　#<data>, <ea>

属性：　サイズ＝(バイト、ロング・ワード)

説明：　デスティネーション・オペランドの1ビットをテストし、そのビットの状態をコンディション・コードZに反映します。データ・レジスタがデスティネーションの場合、モジュロ32のビット番号で32ビットのうちの任意のビットを指定することができます。メモリ・ロケーションがデスティネーションのときは、操作はバイト操作となり、ビット番号はモジュロ8となります。いずれの場合にも、ビット0が最下位ビットです。この操作のビット番号は次の2とおりの方法で指定できます。

1. イミディエイト：ビット番号を命令の第2ワードで指定する。
2. レジスタ：ビット番号は命令で指定するデータ・レジスタにある。

コンディション・コード：

X	N	Z	V	C
—	—	*	—	—

X－影響を受けない。
N－影響を受けない。
Z－テストされたビットが0であればセット、それ以外のときはクリア。
V－影響を受けない。
C－影響を受けない。

命令フォーマット(ビット番号ダイナミック、レジスタで指定)：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
										実効アドレス					
0	0	0	0	レジスタ			1	0	0	モード			レジスタ		

命令フィールド(ビット番号ダイナミック)：

レジスタ・フィールド――ビット番号をもつデータ・レジスタを指定。

実効アドレス・フィールド――デスティネーション・ロケーションを指定、次に示すとおり、データ・アドレッシング・モードのみ可。

ｱﾄﾞﾚｯｼﾝｸﾞ･ﾓｰﾄﾞ	モード	レジスタ
Dn*	000	reg. number : Dn
An	－	－
(An)	010	reg. number : An
(An)＋	011	reg. number : An
－(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

ｱﾄﾞﾚｯｼﾝｸﾞ･ﾓｰﾄﾞ	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

＊ロング・ワードのみ、その他はすべてバイトのみ。

命令フォーマット(ビット番号スタティック、イミディエイト・データとして指定)：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	0	0	実効アドレス モード			実効アドレス レジスタ		
0	0	0	0	0	0	0	0	ビット番号							

命令フィールド(ビット番号スタティック)：

実効アドレス・フィールド――デスティネーション・ロケーションを指定。次のとおり、データ・アドレッシング・モードのみ可。

ｱﾄﾞﾚｯｼﾝｸﾞ･ﾓｰﾄﾞ	モード	レジスタ
Dn*	000	reg. number : Dn
An	－	－
(An)	010	reg. number : An
(An)＋	011	reg. number : An
－(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

ｱﾄﾞﾚｯｼﾝｸﾞ･ﾓｰﾄﾞ	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	－	－
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

ビット番号フィールド――ビット番号を指定。

CAS / CAS2

Compare and Swap with Operand ・オペランドとの比較および交換

操作：　CAS デスティネーション――比較オペランド→cc；
Z＝1であれば、更新オペランド→デスティネーション
そうでなければ、デスティネーション→比較オペランド
CAS2 デスティネーション1――比較オペランド1→cc；
Z＝1であれば、デスティネーション2－比較オペランド2→cc；
Z＝1であれば、更新オペランド1→デスティネーション1
更新オペランド2→デスティネーション2
そうでなければ、デスティネーション1→比較オペランド1
デスティネーション2→比較オペランド2

アセンブラ・シンタックス：　CAS Dc, Du, <ea>
CAS2 Dc1：Dc2,Du1：Du2, (Rn1)：(Rn2)

属性：　サイズ＝(バイト＊、ワード、ロング・ワード)

説明：　CAS命令は実効アドレス・オペランドを比較オペランド(Dc)と比較します。両方のオペランドの値が等しい場合は、更新オペランド(Du)を実効アドレス・オペランドに書き込みます。そうでない場合は、実効アドレス・オペランドを比較オペランド(Dc)に書き込みます。

CAS2命令はメモリ・オペランド1(Rn1)を比較オペランド1(Dc1)と比較します。両方のオペランドの値が等しい場合は、メモリ・オペランド2(Rn2)を比較オペランド2(Dc2)と比較します。なおも両方のオペランドが等しい場合は、更新オペランド(Du1およびDu2)をメモリ・オペランド(Rn1およびRn2)に書き込みます。このいずれかの比較が一致しなかった場合は、メモリ・オペランド(Rn1およびRn2)を比較オペランド(Dc1およびDc2)に書き込みます。

いずれの操作ともリード・モディファイ・ライト・サイクルを使用してメモリにアクセスしますので、これらの命令を中断することはできません。これにより、複数のプロセッサ間の同期をとることができます。「3.4　CASおよびCAS2命令の使用法」に、これらの命令の代表的な応用例が掲載されていますので参照してください。

コンディション・コード：

X	N	Z	V	C
—	*	*	*	*

X－影響を受けない。
N－結果が負であればセット、それ以外のときはクリア。
Z－結果が0であればセット、それ以外のときはクリア。
V－オーバフローが発生すればセット、それ以外のときはクリア。
C－キャリが発生すればセット、それ以外のときはクリア。

＊CAS2ではバイト・オペランドは使用できない。

命令フォーマット(CAS)：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	サイズ		0	1	1	実効アドレス					
										モード			レジスタ		
0	0	0	0	0	0	0	Du			0	0	0	Dc		

命令フィールド：

サイズ・フィールド――操作サイズを指定。

01－バイト操作

10－ワード操作

11－ロング・ワード操作

実効アドレス・フィールド――メモリ・オペランドのロケーションを指定。以下に示すように、メモリ可変アドレッシング・モードのみ可。

ｱﾄﾞﾚｯｼﾝｸﾞ･ﾓｰﾄﾞ	モード	レジスタ
Dn	–	–
An	–	–
(An)	010	reg. number : An
(An)+	011	reg. number : An
–(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

ｱﾄﾞﾚｯｼﾝｸﾞ･ﾓｰﾄﾞ	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	–	–
(d_{16},PC)	–	–
(d_8,PC,Xn)	–	–
(bd,PC,Xn)	–	–
([bd,PC,Xn],od)	–	–
([bd,PC],Xn,od)	–	–

Du フィールド――比較結果が一致した場合、メモリ・オペランドのロケーションへ書き込まれる更新値をもつデータ・レジスタを指定。

Dc フィールド――メモリ・オペランドと比較されるテスト値をもつデータ・レジスタを指定。

命令フォーマット(CAS2)：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	サイズ		0	1	1	1	1	1	1	0	0
D/A1	Rn1			0	0	0	Du1			0	0	0	Dc1		
D/A2	Rn2			0	0	0	Du2			0	0	0	Dc2		

命令フィールド：

サイズ・フィールド――操作サイズを指定。

10－ワード操作

11－ロング・ワード操作

D/A1、D/A2フィールド――Rn1およびRn2がそれぞれデータ・レジスタまたはアドレス・レジスタのいずれを参照するかを指定。

0－対応するレジスタはデータ・レジスタ。

1－対応するレジスタはアドレス・レジスタ。

Rn1、Rn2フィールド――第1および第2のメモリ・オペランドのアドレスをもつレジスタ番号をそれぞれ指定。メモリ内でオペランドがオーバラップしている場合、メモリの更新結果は不定となる。

Du1、Du2フィールド――比較が一致した場合に第1および第2のメモリ・オペランド・ロケーションに書き込む更新値をもつデータ・レジスタを指定する。

Dc1、Dc2フィールド――それぞれ第1および第2のメモリ・オペランドと比較するテスト値をもつデータ・レジスタを指定。Dc1およびDc2が同じデータ・レジスタを指定していて、比較結果が一致しなかった場合、データ・レジスタには第1メモリ・オペランドの値が格納される。

プログラミング上の注意：

CASおよびCAS2命令は、マルチプロセッシング環境でのシステム制御データ構造の更新操作を、安全に実行するために使用できます。

CHK

Check Register Against Bounds・レジスタ境界チェック

操作：　if Dn <0またはDn>ソース then TRAP

アセンブラ・シンタックス：　CHK <ea>, Dn

属性：　サイズ＝(ワード、ロング・ワード)

説明：　命令で指定するデータ・レジスタの値をゼロおよび上限値(実効アドレス・オペランド)と比較します。上限値は2の補数の整数です。レジスタ値が0より小さいかまたは上限値より大きい場合は、ベクタ番号6のCHK命令例外が発生します。

コンディション・コード：

X	N	Z	V	C
—	*	U	U	U

X－影響を受けない。

N－Dn<0であればセット、Dn>ソースであればクリア。それ以外のときは不定。

Z－不定

V－不定

C－不定

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
										実効アドレス					
0	1	0	0	レジスタ			サイズ		0	モード			レジスタ		

命令フィールド：

レジスタ・フィールド——チェックする値をもつデータ・レジスタを指定。

サイズ・フィールド——操作サイズを指定。

11－ワード操作

10－ロング・ワード操作

実効アドレス・フィールド——上限値オペランドを指定。次に示すとおり、データ・アドレッシング・モードのみ可。

ｱﾄﾞﾚｯｼﾝｸﾞ・ﾓｰﾄﾞ	モード	レジスタ
Dn	000	reg. number : Dn
An	–	–
(An)	010	reg. number : An
(An)+	011	reg. number : An
−(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

ｱﾄﾞﾚｯｼﾝｸﾞ・ﾓｰﾄﾞ	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

CHK2

Check Register Against Bounds • レジスタ境界チェック

操作：　if　Rn＜ソース下限値　または
　　Rn＞ソース上限値
　Then　TRAP

アセンブラ・シンタックス：　CHK2　＜ea＞,Rn

属性：　サイズ＝(バイト、ワード、ロング・ワード)

説明：　Rnの値を上限および下限値と比較します。実効アドレスには上限および下限値のペアがあり、下限値のあとに上限値がきます。符号付き比較の場合は、算術的に小さいほうの値を下限値として使用しなければなりません。符号なし比較の場合は、論理的に小さいほうの値が下限値でなければなりません。

データおよび境界のサイズは、バイト、ワード、またはロング・ワードとして指定することができます。Rnがデータ・レジスタであって、オペランド・サイズがバイトまたはワードの場合は、Rnで該当する下位部分だけがチェックされます。Rnがアドレス・レジスタであって、操作サイズがバイトまたはワードの場合、境界オペランドは32ビットに符号拡張され、その結果のオペランドがAnの全32ビットと比較されます。

上限値と下限値が等しい場合、有効範囲は単独値になります。レジスタ値が上限または下限値を超えた場合、ベクタ番号6のCHK命令例外が発生します。

コンディション・コード：

X	N	Z	V	C
—	U	*	U	*

X－影響を受けない。

N－不定

Z－Rnがいずれかの限界値に等しい場合セット、それ以外のときはクリア。

V－不定

C－Rnが限界値を超えている場合セット、それ以外のときはクリア。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	サイズ		0	1	1	実効アドレス					
										モード			レジスタ		
D/A	レジスタ			1	0	0	0	0	0	0	0	0	0	0	0

命令フィールド：

サイズ・フィールド――操作サイズを指定。

00－バイト操作

01－ワード操作

10－ロング・ワード操作

実効アドレス・フィールド――境界オペランドのロケーションを指定。以下に示すように制御アドレッシング・モードのみ可。

ｱﾄﾞﾚｯｼﾝｸﾞ･ﾓｰﾄﾞ	モード	レジスタ
Dn	–	–
An	–	–
(An)	010	reg. number : An
(An)＋	–	–
－(An)	–	–
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

ｱﾄﾞﾚｯｼﾝｸﾞ･ﾓｰﾄﾞ	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	–	–
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

D/Aフィールド――チェックするレジスタがデータ・レジスタであるかアドレス・レジスタであるかを指定。

0－データ・レジスタ

1－アドレス・レジスタ

レジスタ・フィールド――チェックするアドレス・レジスタまたはデータ・レジスタを指定。

CLR

Clear an Operand • オペランドのクリア

操作：　0→デスティネーション

アセンブラ・シンタックス：　CLR < ea >

属性：　サイズ＝(バイト、ワード、ロング・ワード)

説明：　デスティネーション・オペランドを0にクリアします。操作サイズはバイト、ワード、ロング・ワードが指定できます。

コンディション・コード：

X	N	Z	V	C
—	0	1	0	0

X－影響を受けない。
N－常にクリア。
Z－常にセット。
V－常にクリア。
C－常にクリア。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	1	0	サイズ		実効アドレス					
										モード			レジスタ		

命令フィールド：

サイズ・フィールド——操作サイズを指定。
　00－バイト操作
　01－ワード操作
　10－ロング・ワード操作

実効アドレス・フィールド——デスティネーション・ロケーションを指定。次に示すとおり、データ可変アドレッシング・モードのみ可。

ｱﾄﾞﾚｯｼﾝｸﾞ・ﾓｰﾄﾞ	モード	レジスタ
Dn	000	reg. number : Dn
An	−	−
(An)	010	reg. number : An
(An)＋	011	reg. number : An
−(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

ｱﾄﾞﾚｯｼﾝｸﾞ・ﾓｰﾄﾞ	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	−	−
(d_{16},PC)	−	−
(d_8,PC,Xn)	−	−
(bd,PC,Xn)	−	−
([bd,PC,Xn],od)	−	−
([bd,PC],Xn,od)	−	−

CMP

Compare • 比較

操作：　デスティネーション──ソース→cc

アセンブラ・シンタックス：　CMP < ea >, Dn

属性：　サイズ＝(バイト、ワード、ロング・ワード)

説明：　ソース・オペランドをデスティネーション・データ・レジスタから減算し、その結果に従ってコンディション・コードをセットします。データ・レジスタの内容は変化しません。操作サイズはバイト、ワード、ロング・ワードが指定できます。

コンディション・コード：

X	N	Z	V	C
—	*	*	*	*

X－影響を受けない。

N－結果が負であればセット、それ以外のときはクリア。

Z－結果が0であればセット、それ以外のときはクリア。

V－オーバフローが発生すればセット、それ以外のときはクリア。

C－ボローが発生すればセット、それ以外のときはクリア。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	レジスタ			Opモード			実効アドレス					
										モード			レジスタ		

命令フィールド：

レジスタ・フィールド──デスティネーションのデータ・レジスタを指定。

Opモード・フィールド：

バイト	ワード	ロング・ワード	操　作
000	001	010	(< Dn >) － (< ea >)

実効アドレス・フィールド――ソース・オペランドを指定。次に示すとおり、すべてのアドレッシング・モードが可。

アドレッシング・モード	モード	レジスタ
Dn	000	reg. number : Dn
An*	001	reg. number : An
(An)	010	reg. number : An
(An)+	011	reg. number : An
−(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

アドレッシング・モード	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*ワードとロング・ワードのみ。

注：デスティネーションがアドレス・レジスタのときはCMPAを使用し、ソースがイミディエイト・データのときはCMPIを使用します。またメモリとメモリの比較にはCMPMを使用します。ほとんどのアセンブラは自動的にこの区別を行ないます。

CMPA

Compare Address・アドレス比較

操作：　デスティネーション−ソース

アセンブラ・シンタックス：　CMPA < ea >, An

属性：　サイズ=(ワード、ロング・ワード)

説明：　ソース・オペランドをデスティネーションのアドレス・レジスタから減算し、その結果に従ってコンディション・コードをセットします、アドレス・レジスタの内容は変化しません。操作サイズは、ワードまたはロング・ワードが指定できます。ワード長のソース・オペランドは、操作を実行する前に32ビットに符号拡張されます。

コンディション・コード：

X	N	Z	V	C
—	*	*	*	*

X−影響を受けない。
N−結果が負であればセット、それ以外のときはクリア。
Z−結果が0であればセット、それ以外のときはクリア。
V−オーバフローが発生すればセット、それ以外のときはクリア。
C−ボローが発生すればセット、それ以外のときはクリア。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	レジスタ			Opモード			実効アドレス					
										モード			レジスタ		

命令フィールド：

レジスタ・フィールド——デスティネーションのアドレス・レジスタを指定。

Opモード・フィールド——操作サイズを指定。

011 −ワード操作。ソース・オペランドはロング・ワードに符号拡張され、アドレス・レジスタの全32ビットを使用して操作を実行。

111 −ロング・ワード操作

実効アドレス・フィールド——ソース・オペランドを指定。次に示すとおり、すべてのアドレッシング・モードが可。

ｱﾄﾞﾚｯｼﾝｸﾞ･ﾓｰﾄﾞ	モード	レジスタ
Dn	000	reg. number : Dn
An	001	reg. number : An
(An)	010	reg. number : An
(An)+	011	reg. number : An
−(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

ｱﾄﾞﾚｯｼﾝｸﾞ･ﾓｰﾄﾞ	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

CMPI

Compare Immediate・イミディエイト比較

操作：　デスティネーション－イミディエイト・データ

アセンブラ・シンタックス：　CMPI #<data>, <ea>

属性：　サイズ＝(バイト、ワード、ロング・ワード)

説明：　イミディエイト・データをデスティネーション・オペランドから減算し、その結果に従ってコンディション・コードをセットします。デスティネーション・ロケーションの内容は変化しません。操作サイズは、バイト、ワードまたはロング・ワードが指定できます。イミディエイト・データのサイズは操作サイズと同じです。

コンディション・コード：

X	N	Z	V	C
—	*	*	*	*

X－影響を受けない。
N－結果が負であればセット、それ以外のときはクリア。
Z－結果が0であればセット、それ以外のときはクリア。
V－オーバフローが発生すればセット、それ以外のときはクリア。
C－ボローが発生すればセット、それ以外のときはクリア。

命令フォーマット：

<table>
<tr><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">1</td><td rowspan="2">1</td><td rowspan="2">0</td><td rowspan="2">0</td><td colspan="2" rowspan="2">サイズ</td><td colspan="6">実効アドレス</td></tr>
<tr><td colspan="3">モード</td><td colspan="3">レジスタ</td></tr>
<tr><td colspan="8">ワード・データ（16ビット）</td><td colspan="8">バイト・データ（8ビット）</td></tr>
<tr><td colspan="16">ロング・データ（32ビット）</td></tr>
</table>

命令フィールド：

サイズ・フィールド――操作サイズを指定。
　00－バイト操作
　01－ワード操作
　10－ロング・ワード操作

実効アドレス・フィールド——デスティネーションのオペランドを指定。次に示すとおり、データ・アドレッシング・モードのみ可。

ｱﾄﾞﾚｯｼﾝｸﾞ･ﾓｰﾄﾞ	モード	レジスタ
Dn	000	reg. number : Dn
An	–	–
(An)	010	reg. number : An
(An)＋	011	reg. number : An
−(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

ｱﾄﾞﾚｯｼﾝｸﾞ･ﾓｰﾄﾞ	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	–	–
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

イミディエイト・フィールド——（命令直後のデータ）

サイズ＝00　データはイミディエイト・ワードの下位バイト。

サイズ＝01　データはイミディエイト・ワード全体。

サイズ＝10　データは次の2つのイミディエイト・ワード。

CMPM

Compare Memory・メモリ比較

操作：　デスティネーション――ソース→cc

アセンブラ・シンタックス：　CMPM (Ay)+, (Ax)+

属性：　サイズ＝(バイト、ワード、ロング・ワード)

説明：　ソース・オペランドをデスティネーション・オペランドから減算し、その結果に従って、コンディション・コードをセットします。デスティネーション・ロケーションの内容は変化しません。オペランドは、命令で指定されているアドレス・レジスタを使用して、常にポストインクリメント・アドレッシング・モードでアドレス指定されます。操作サイズはバイト、ワード、ロング・ワードが指定できます。

コンディション・コード：

X	N	Z	V	C
—	*	*	*	*

X－影響を受けない。

N－結果が負であればセット、それ以外のときはクリア。

Z－結果が0であればセット、それ以外のときはクリア。

V－オーバフローが発生すればセット、それ以外のときはクリア。

C－ボローが発生すればセット、それ以外のときはクリア。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	レジスタ Ax			1	サイズ		0	0	1	レジスタ Ay		

命令フィールド：

レジスタAxフィールド――(常にデスティネーション) ポストインクリメント・アドレッシング・モードで使用するアドレス・レジスタを指定。

サイズ・フィールド――操作サイズを指定。

00－バイト操作

01－ワード操作

10－ロング・ワード

レジスタAyフィールド――(常にソース・オペランド)ポストインクリメント・アドレッシング・モードに使用するアドレス・レジスタを指定。

CMP2

Compare Register Against Bounds・レジスタ境界チェック

操作：　Rn < ソース　下限値　または

Rn > ソース　上限値

の比較を行ない、その結果によってコンディション・コードをセットする。

アセンブラ・シンタックス：　CMP2　< ea >, Rn

属性：　サイズ＝(バイト、ワード、ロング・ワード)

説明：　Rnの値を上限および下限値と比較します。実効アドレスには、上限および下限値のペアがあり、下限値のあとに上限値が続きます。符号付き比較の場合、算術的に小さいほうの値を下限値として使用します。符号なし比較の場合は、論理的に小さいほうの値を下限値にしなければなりません。

データおよび境界のサイズは、バイト、ワード、またはロング・ワードで指定することができます。Rnがデータ・レジスタであって、オペランド・サイズがバイトまたはワードの場合は、Rnで該当する下位部分だけがチェックされます。Rnがアドレス・レジスタであって、操作サイズがバイトまたはワードの場合は、境界オペランドは32ビットに符号拡張され、その結果のオペランドがAnの全32ビットと比較されます。

上限値と下限値が等しい場合、有効範囲は単独値になります。

注：この命令は、境界値を超えたときに例外処理を行なわずコンディション・コードをセットする点を除いてCHK2と同じです。

コンディション・コード：

X	N	Z	V	C
—	U	*	U	*

X－影響を受けない。

N－不定

Z－Rnがいずれかの限界値に等しい場合セット、それ以外のときはクリア。

V－不定

C－Rnが限界値を超えている場合セット、それ以外のときはクリア。

命令フォーマット：

<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">0</td><td colspan="2" rowspan="2">サイズ</td><td rowspan="2">0</td><td rowspan="2">1</td><td rowspan="2">1</td><td colspan="6">実効アドレス</td></tr>
<tr><td colspan="3">モード</td><td colspan="3">レジスタ</td></tr>
<tr><td>D/A</td><td colspan="3">レジスタ</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>

命令フィールド：

サイズ・フィールド――操作サイズを指定。

00－バイト操作

01－ワード操作

10－ロング・ワード

実効アドレス・フィールド――限界値ペアのロケーションを指定。以下に示すように制御アドレッシング・モードのみ可。

アドレッシング・モード	モード	レジスタ
Dn	－	－
An	－	－
(An)	010	reg. number : An
(An)＋	－	－
－(An)	－	－
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

アドレッシング・モード	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	－	－
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

D/Aフィールド――チェックするレジスタがデータ・レジスタであるかアドレス・レジスタであるかを指定。

0－データ・レジスタ

1－アドレス・レジスタ

レジスタ・フィールド――チェックするアドレス・レジスタまたはデータ・レジスタを指定。

cpBcc

Branch on Coprocessor Condition・コプロセッサ条件による分岐

操作：　If cpcc＝真 then スキャンPC＋d→PC

アセンブラ・シンタックス：　cpBcc ＜label＞

属性：　サイズ＝(ワード、ロング・ワード)

説明：　指定されたコプロセッサの条件が真の場合、プログラムの実行を、スキャンPCのロケーション＋ディスプレースメントから続行します。スキャンPCの値は最初のディスプレースメント・ワードのアドレスです。ディスプレースメントは、スキャンPCからデスティネーションPCまでの相対的な距離をバイト単位で示す2の補数の整数です。ディスプレースメントは、16ビットまたは32ビットのいずれかが指定できます。コプロセッサは、オペレーション・ワードの中のコンディション・フィールドから、特定の条件を決定します。詳細については、「10. 4. 1　走査用PC」を参照してください。

コンディション・コード：　影響を受けない。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	CP-ID ≠000			0	1	サイズ	コプロセッサ条件					
オプションのコプロセッサ定義拡張ワード															
ワードまたは															
ロング・ワード・ディスプレースメント															

命令フィールド：

Cp-Idフィールド――この操作を実行するコプロセッサを識別。Cp-Idが000のときはFライン例外が発生する。

サイズ・フィールド――ディスプレースメントのサイズを指定。

0－16ビット・ディスプレースメント

1－32ビット・ディスプレースメント

コプロセッサ条件フィールド――テストするコプロセッサ条件を指定。このフィールドは、コプロセッサに渡される。コプロセッサは、メイン・プロセッサに対してこの命令を処理するための指令を与える。

16ビット・ディスプレースメント・フィールド――16ビットのディスプレースメント値。

32ビット・ディスプレースメント・フィールド――32ビットのディスプレースメント値。

Test Coprocessor Condition Decrement and Branch

コプロセッサ条件テスト、デクリメントおよび分岐

操作：　If cpcc ＝偽　then (Dn －1→Dn；If Dn≠－1 then scanPC＋d→PC)

アセンブラ・シンタックス：　cpDBcc Dn,<label>

属性：　サイズ＝(ワード)

説明：　指定されたコプロセッサの条件が真の場合は、次の命令の実行に移ります。それ以外の場合は、指定されたデータ・レジスタの下位ワードが1だけデクリメントされます。その結果が－1に等しい場合は、次の命令の実行に移ります。結果が－1に等しくない場合、プログラムの実行をスキャンPC＋符号拡張された16ビット・ディスプレースメントによって示されるロケーションから続行します。スキャンPCの値は、ディスプレースメント・ワードのアドレスです。ディスプレースメントは、スキャンPCからデスティネーションPCまでの相対的な距離をバイト単位で示す2の補数の整数です。コプロセッサは、オペレーション・ワードに続くコンディション・ワードから、特定の条件を決定します。詳細については、「10. 4. 1　走査用PC」を参照してください。

コンディション・コード：　影響を受けない。

命令フォーマット：

<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td colspan="3">CP-ID ≠000</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td colspan="3">レジスタ</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td colspan="6">コプロセッサ条件</td></tr>
<tr><td colspan="16">オプションのコプロセッサ定義拡張ワード</td></tr>
<tr><td colspan="16">ディスプレースメント(16ビット)</td></tr>
</table>

命令フィールド：

Cp-Id フィールド――この操作の対象となるコプロセッサを識別。Cp-Idが000のときには、Fライン例外が発生する。

レジスタ・フィールド――カウンタとして使用するデータ・レジスタを指定。

コプロセッサ条件フィールド――テストされるべきコプロセッサ条件を指定。このフィールドはコプロセッサに渡される。コプロセッサはこの命令を処理するためにメイン・プロセッサに対して指令を与える。

ディスプレースメント・フィールド――分岐の距離(バイト単位)を指定。

cpGEN

Coprocessor General Function・コプロセッサの一般機能

操作：　コマンド・ワードをコプロセッサに渡す。

アセンブラ・シンタックス：　cpGEN <コプロセッサによって定義されているパラメータ>

属性：　サイズなし

説明：　オペレーション・ワードに続くコマンド・ワードを指定されたコプロセッサに転送します。コプロセッサはコマンド・ワードから、指定された操作を決定します。通常コプロセッサはこの命令の細かい使用法を定義してこの命令セットを提供します。

コンディション・コード：　コプロセッサによって変更される場合がある。それ以外のときは不変。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	CP-ID ≠000			0	0	0	実効アドレス モード			実効アドレス レジスタ		
コプロセッサ依存コマンド・ワード															
オプションの実効アドレスまたはコプロセッサ定義拡張ワード															

命令フィールド：

Cp-Id フィールド――この操作の対象となるコプロセッサを識別。Cp-Idが000のときには、Fライン例外が発生する。

実効アドレス・フィールド――コプロセッサの外部の任意のオペランドのロケーションを指定、使用可能なアドレッシング・モードは、実行される操作ごとに決められている。

コプロセッサ・コマンド・フィールド――実行されるべきコプロセッサ操作を指定、このワードはコプロセッサに渡される。コプロセッサはこの命令を処理するためにメイン・プロセッサに対して指令を与える。

cpRESTORE

Coprocessor Restore Functions(Privileged Instruction)・コプロセッサ・リストア機能(特権命令)

操作：　スーパバイザ状態ではコプロセッサの内部状態をリストア
ユーザ状態ではTRAP

アセンブラ・シンタックス：　cpRESTORE　<ea>

属性：　サイズなし

説明：　コプロセッサの内部状態をリストアします。通常この状態はcpSAVE命令でセーブされたものです。

コンディション・コード：　影響を受けない。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	CP-ID ≠000			1	0	1	実効アドレス					
										モード			レジスタ		

命令フィールド：

Cp-Id フィールド——リストアするコプロセッサを識別。Cp-Idが000のときにはFライン例外が発生する。

実効アドレス・フィールド——コプロセッサの内部状態が格納されているロケーションを指定。次に示すとおり、ポストインクリメントまたは制御アドレッシング・モードのみ可。

ｱﾄﾞﾚｯｼﾝｸﾞ・ﾓｰﾄﾞ	モード	レジスタ
Dn	−	−
An	−	−
(An)	010	reg. number : An
(An)+	011	reg. number : An
−(An)	−	−
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

ｱﾄﾞﾚｯｼﾝｸﾞ・ﾓｰﾄﾞ	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	−	−
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

注：コプロセッサから返されるフォーマット・ワードが“come again”となっていた場合、保留中の割込みはサービスされません。

cpSAVE

Coprocessor Save Function(Privileged Instruction)・

コプロセッサ・セーブ機能(特権命令)

操作：　スーパバイザ状態ではコプロセッサの内部状態をリストア
ユーザ状態ではTRAP

アセンブラ・シンタックス：　cpSAVE <ea>

属性：　サイズなし

説明：　この命令はcpRESTORE命令でリストアできるフォーマットで、コプロセッサの内部状態をセーブするために使用されます。

コンディション・コード：　影響を受けない。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
										実効アドレス					
										モード			レジスタ		
1	1	1	1	CP-ID ≠000			1	0	0						

命令フィールド：

Cp-Id フィールド――この操作の対象となるコプロセッサを識別。Cp-Idが000のときには、Fライン例外が発生する。

実効アドレス・フィールド――コプロセッサの内部状態をセーブするロケーションを指定。次に示すとおり、プリデクリメントまたは可変制御アドレッシング・モードのみ可。

ｱﾄﾞﾚｯｼﾝｸﾞ・ﾓｰﾄﾞ	モード	レジスタ
Dn	−	−
An	−	−
(An)	010	reg. number : An
(An)＋	−	−
−(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

ｱﾄﾞﾚｯｼﾝｸﾞ・ﾓｰﾄﾞ	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	−	−
(d_{16},PC)	−	−
(d_8,PC,Xn)	−	−
(bd,PC,Xn)	−	−
([bd,PC,Xn],od)	−	−
([bd,PC],Xn,od)	−	−

cpScc

Set on Coprocessor Condition・コプロセッサ条件によるセット

操作：　If cpcc =真 then 1s→デスティネーション
else 0s →デスティネーション

アセンブラ・シンタックス：　cpScc <ea>

属性：　サイズ=(バイト)

説明：　指定されたコプロセッサのコンディション・コードをテストします。その条件が真であれば実効アドレスで指定されるバイトがTRUE(全ビット1)に設定され、条件が偽であった場合は、そのバイトがFALSE(全ビット0)に設定されます。コプロセッサはオペレーション・ワードの次の条件ワードから、特定の条件を決定します。

コンディション・コード：　影響を受けない。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
										実効アドレス					
1	1	1	1	CP-ID ≠000			0	0	1	モード			レジスタ		
0	0	0	0	0	0	0	0	0	0	コプロセッサ条件					
オプションの実効アドレスまたはコプロセッサ定義拡張ワード															

命令フィールド：

Cp-Id フィールド——この操作の対象となるコプロセッサを識別。Cp-Idが000のときには、Fライン例外が発生する。

実効アドレス——デスティネーションのロケーションを指定。次に示すとおり、データ可変アドレッシング・モードのみ可。

アドレッシング・モード	モード	レジスタ
Dn	000	reg. number : Dn
An	−	−
(An)	010	reg. number : An
(An)+	011	reg. number : An
−(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

アドレッシング・モード	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	−	−
(d_{16},PC)	−	−
(d_8,PC,Xn)	−	−
(bd,PC,Xn)	−	−
([bd,PC,Xn],od)	−	−
([bd,PC],Xn,od)	−	−

コプロセッサ条件フィールド——テストするコプロセッサ条件を指定、このフィールドはコプロセッサに渡され、さらにメイン・プロセッサに対してこの命令を処理するための指令が与えられる。

cpTRAPcc

Trap on Coprocessor Condition • コプロセッサ条件によるトラップ

操作：　If cpcc ＝真 then TRAP

アセンブラ・シンタックス：　cpTRAPcc

cpTRAPcc #< data >

属性：　サイズなし、またはサイズ＝(ワード、ロング・ワード)

説明：　指定されたコプロセッサの条件コードをテストします。選択されたコプロセッサ条件が真の場合、プロセッサはベクタ番号7のcpTRAPcc例外の処理を開始します。スタックに置かれるプログラム・カウンタ値は、次の命令のアドレスです。選択された条件が真でなかった場合は、何も実行せず、次の命令の実行に移ります。コプロセッサはオペレーション・ワードの次の条件ワードから、特定の条件を決定します。条件ワードの次に、イミディエイト・データとして指定されるユーザ定義のデータ・オペランドがあり、トラップ・ハンドラがこれを使用します。

コンディション・コード：　影響を受けない。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	CP-ID ≠000			0	0	1	1	1	1	Opモード		
0	0	0	0	0	0	0	0	0	0	コプロセッサ条件					
オプションのコプロセッサ定義拡張ワード															
オプションのワード															
またはロング・ワード・オペランド															

命令フィールド：

Cp-Id フィールド――この操作の対象となるコプロセッサを識別。Cp-Idが000のときには、Fライン例外が発生する。

Opモード・フィールド――命令フォームを選択。

010 －命令のあとに1オペランド・ワードが続く。

011 －命令のあとに2オペランド・ワードが続く。

100 －命令のあとにはオペランド・ワードはない。

コプロセッサ条件フィールド――テストするコプロセッサ条件を指定、このフィールドはコプロセッサに渡され、さらにメイン・プロセッサに対してこの命令を処理するための指令が与えられる。

DBcc

Test Condition、*Decrement*、*and Branch*・条件テスト、デクリメントおよび分岐

操作：　If　条件が偽 then(Dn − 1 → Dn；
If　Dn ≠ − 1　then　PC + d → PC)

アセンブラ・シンタックス：　DBcc Dn, < label >

属性：　サイズ(ワード)

説明：　命令のループを制御します。パラメータは、コンディション・コード、データ・レジス(カウンタ)、およびディスプレースメント値の3つです。この命令は最初に条件(ループ終了)をテストします。それが真の場合は何もしません。終了条件が真でない場合は、カウンタ・データ・レジスタの下位16ビットが1だけデクリメントされます。その結果が−1の場合は次の命令に実行が移ります。結果が−1でない場合は、現在のPCの値に符号拡張された16ビットのディスプレースメントを加えたロケーションから実行を続行します。PCの値はDBcc命令＋2つの命令ワードのアドレスです。ディスプレースメントは現在のPCからデスティネーションPCの相対距離を示す2の補数です。
コンディション・コードccは、次の条件の1つを指定します。

CC	キャリ・クリア	0100	$\overline{C}$	LS	ローか同じ	0011	$C + Z$
CS	キャリ・セット	0101	C	LT	より小さい	1101	$N \cdot \overline{V} + \overline{N} \cdot V$
EQ	等しい	0111	Z	MI	マイナス	1011	N
F	等しくない	0001	0	NE	等しくない	0110	$\overline{Z}$
GE	大きいか等しい	1100	$N \cdot V + \overline{N} \cdot \overline{V}$	PL	プラス	1010	$\overline{N}$
GT	より大きい	1110	$N \cdot V \cdot \overline{Z} + \overline{N} \cdot \overline{V} \cdot \overline{Z}$	T	常に真	0000	1
HI	ハイ	0010	$\overline{C} \cdot \overline{Z}$	VC	ｵｰﾊﾞﾌﾛｰ・ｸﾘｱ	1000	$\overline{V}$
LE	小さいか等しい	1111	$Z + N \cdot \overline{V} + \overline{N} \cdot V$	VS	ｵｰﾊﾞﾌﾛｰ・ｾｯﾄ	1001	V

コンディション・コード：　影響を受けない。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	コンディション				1	1	0	0	1	レジスタ		
ディスプレースメント（16ビット）															

命令フィールド：

コンディション・フィールド——上記条件のうち1つの2進コードを指定。
レジスタ・フィールド——カウンタとして使用するデータ・レジスタを指定。
ディスプレースメント・フィールド——分岐の距離をバイト数で指定。

注：1. 終了条件は高水準言語のUNTIL ループ構造とよく似ています。たとえば、DBMIは“マイナスになるまでデクリメントし分岐する”と表現できます。

2. ほとんどのアセランブラはループを終了させるのがカウントだけのときには、DBFの代わりにDBRAを使用することができます(どの条件もテストしない)。

3. プログラムは、ループの先頭からまたはループの最後尾にあるDBcc命令に分岐することによってループに入ります。インデックス・アドレッシング・モードおよびダイナミック指定ビット操作に対しては、ループの先頭から入るほうがよいでしょう。この場合、制御インデックス・カウントは実行したいループ回数より1少ない数になります。しかし、最後尾のDBcc命令に直接分岐してループに入るときは、制御インデックスはループ実行カウントと同じでなければなりません。この場合、カウントがゼロになるとDBccは分岐しないため、メイン・ループは実行されません。

DIVS / DIVSL

Signed Divide・符号付き除算

操作：　デスティネーション/ソース→デスティネーション

アセンブラ・シンタックス：
DIVS.W <ea>,Dn　32/16→16r：16q
DIVS.L <ea>,Dq　32/32→32q
DIVS.L <ea>,Dr：Dq　64/32→32r：32q
DIVSL.L <ea>,Dr：Dq　32/32→32r：32q

属性：　サイズ＝(ワード、ロング・ワード)

説明：　符号付きデスティネーション・オペランドを符号付きソース・オペランドで除算し、符号付きの結果をデスティネーションに格納します。この命令は4種類のフォームの1つを使用します。ワード・フォームは、ロング・ワードをワードで除算します。結果は商が下位ワード(下位16ビット)に入り、余りは結果の上位ワード(上位16ビット)に入ります。余りの符号は被除数の符号と同じです。

第1のロング・ワード・フォームは、ロング・ワードをロング・ワードで除算します。結果はロング・ワードの商で、余りは捨てられます。

第2のロング・フォームは、クワッド・ワード(任意の2個のレジスタ)をロング・ワードで除算します。結果はロング・ワードの商とロング・ワードの余りになります。

第3のロング・フォームは、ロング・ワードをロング・ワードで除算します。結果はロング・ワードの商とロング・ワードの余りになります。

演算中に次の2つの特殊状態が発生することがあります。

1. 0での除算によりトラップが発生します。
2. 命令が完了する前にオーバフローが検出され、フラグがセットされます。命令がオーバフローを検出した場合、コンディション・コードのオーバフロー・フラグをセットするが、オペランドは影響を受けません。

コンディション・コード：

X	N	Z	V	C
—	*	*	*	0

X－影響を受けない。

N－商が負であればセット、それ以外のときはクリア。ただし、オーバフローまたは0での除算が発生した場合は不定。

Z－商が0であればセット、それ以外のときはクリア。ただし、オーバフローまたは0での除算が発生した場合は不定。

V－除算オーバフローが発生するとセットされる。0での除算が発生した場合は不定。それ以外のときはクリア。

C－常にクリア。

命令フォーマット(ワード・フォーム)：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	レジスタ			1	1	1	実効アドレス モード			レジスタ		

命令フィールド：

レジスタ・フィールド――8つのデータ・レジスタのいずれかを指定。このフィールドは、常にデスティネーション・オペランドを指定する。
実効アドレス・フィールド――ソース・オペランドを指定。次に示すとおり、データ・アドレッシング・モードのみ可。

アドレッシング・モード	モード	レジスタ
Dn	000	reg. number : Dn
An	−	−
(An)	010	reg. number : An
(An)+	011	reg. number : An
−(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

アドレッシング・モード	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

注：商が16ビットの符号付き整数より大きいときは、オーバフローが発生します。

命令フォーマット(ロング・フォーム)：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	0	1	実効アドレス モード			レジスタ		
0	レジスタDq			1	サイズ	0	0	0	0	0	0	0	レジスタDr		

命令フィールド：

実効アドレス・フィールド——ソース・オペランドを指定。次に示すとおり、データ・アドレッシング・モードのみ可。

アドレッシング・モード	モード	レジスタ
Dn	000	reg. number：Dn
An	−	−
(An)	010	reg. number：An
(An)＋	011	reg. number：An
−(An)	100	reg. number：An
(d_{16},An)	101	reg. number：An
(d_8,An,Xn)	110	reg. number：An
(bd,An,Xn)	110	reg. number：An
([bd,An,Xn],od)	110	reg. number：An
([bd,An],Xn,od)	110	reg. number：An

アドレッシング・モード	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

レジスタDqフィールド——デスティネーション・オペランドのデータ・レジスタを指定、被除数の下位32ビットがこのレジスタから取り出され、32ビットの商がこのレジスタにロードされる。

サイズ・フィールド——32ビットまたは64ビットの除算のいずれかを選択。

0－32ビットの被除数がレジスタDqにある。

1－64ビットの被除数がDr：Dqにある。

レジスタDrフィールド——除算終了後、このレジスタには32ビットの余りが入れられる。DrとDqが同じレジスタの場合、商だけが返される。サイズが1の場合、このフィールドは被除数の上位32ビットをもっているデータ・レジスタも指定する。

注：商が32ビットの符号付き整数より大きい場合は、オーバフローが発生します。

DIVU / DIVUL

Unsigned Divide • 符号なし除算

操作：　デスティネーション/ソース→デスティネーション

アセンブラ・シンタックス：

DIVU.W < ea >, Dn	32/16→16r : 16q
DIVU.L < ea >, Dq	32/32→32q
DIVU.L < ea >, Dr : Dq	64/32→32r : 32q
DIVUL.L < ea >, Dr : Dq	32/32→32r : 32q

属性：　サイズ＝(ワード、ロング・ワード)

説明：　符号なしデスティネーション・オペランドを符号なしソース・オペランドで除算し、符号なしの結果をデスティネーションに格納します。この命令は4種類のフォームの1つを使用します。ワード・フォームは、ロング・ワードをワードで除算します。結果は商が下位ワード(下位16ビット)に入り、余りは結果の上位ワード(上位16ビット)に入ります。

第1のロング・フォームは、ロング・ワードをロング・ワードで除算します。結果はロング・ワードの商で、余りは捨てられます。

第2のロング・フォームは、クワッド・ワード(任意の2個のレジスタ)をロング・ワードで除算します。結果はロング・ワードの商とロング・ワードの余りになります。

第3のロング・フォームは、ロング・ワードをロング・ワードで除算します。結果はロング・ワードの商とロング・ワードの余りになります。

演算中に次の2つの特殊状態が発生することがあります。

1. 0での除算によりトラップが発生します。
2. 命令が完了する前にオーバフローが検出され、フラグがセットされます。命令がオーバフローを検出した場合、コンディション・コードのオーバフロー・フラグをセットするが、オペランドは影響を受けません。

コンディション・コード：

X	N	Z	V	C
—	*	*	*	0

X－影響を受けない。

N－商が負であればセット、それ以外のときはクリア。ただし、オーバフローまたは0での除算が発生した場合は不定。

Z－商が0であればセット、それ以外のときはクリア。ただし、オーバフローまたは0での除算が発生した場合は不定。

V－除算オーバフローが発生したらセット。0での除算が発生した場合は不定。それ以外のときはクリア。

C－常にクリア。

命令フォーマット（ワード・フォーム）：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	レジスタ			0	1	1	実効アドレス モード			実効アドレス レジスタ		

命令フィールド：

レジスタ・フィールド――8つのデータ・レジスタのいずれかを指定。このフィールドは、常にデスティネーション・オペランドを指定する。

実効アドレス・フィールド――ソース・オペランドを指定。次に示すとおり、データ・アドレッシング・モードのみ可。

アドレッシング・モード	モード	レジスタ
Dn	000	reg. number : Dn
An	−	−
(An)	010	reg. number : An
(An)+	011	reg. number : An
−(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

アドレッシング・モード	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

注：商が16ビットの符号付き整数より大きいときは、オーバフローが発生します。

命令フォーマット（ロング・フォーム）：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	0	1	実効アドレス モード			実効アドレス レジスタ		
0	レジスタDq			0	サイズ	0	0	0	0	0	0	0	レジスタDr		

命令フィールド：

実効アドレス・フィールド――ソース・オペランドを指定。次に示すとおり、データ・アドレッシング・モードのみ可。

アドレッシング・モード	モード	レジスタ
Dn	000	reg. number : Dn
An	–	–
(An)	010	reg. number : An
(An)+	011	reg. number : An
–(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

アドレッシング・モード	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

レジスタDqフィールド――デスティネーション・オペランドのデータ・レジスタを指定、被除数の下位32ビットがこのレジスタから取り出され、32ビットの商がこのレジスタにロードされる。
サイズ・フィールド――32ビットまたは64ビットの除算のいずれかを選択。
　0 – 32ビットの被除数がレジスタDqにある。
　1 – 64ビットの被除数がDr：Dqにある。
レジスタDrフィールド――除算終了後、このレジスタには32ビットの余りが入れられる。DrとDqが同じレジスタの場合、商だけが返される。サイズが1の場合、このフィールドは被除数の上位32ビットをもっているデータ・レジスタも指定する。

注：商が32ビットの符号なし整数より大きいときは、オーバフローが発生します。

EOR

Exclusive OR Logical・排他的論理和

操作：　ソース⊕デスティネーション→デスティネーション
アセンブラ・シンタックス：　EOR Dn, < ea >
属性：　サイズ＝(バイト、ワード、ロング・ワード)
説明：　ソース・オペランドとデスティネーション・オペランドとの排他的論理和をとり、結果をデスティネーション・ロケーションに格納します。操作サイズはバイト、ワード、ロング・ワードが指定できます。この操作ではソース・オペランドは、データ・レジスタに限定されます。デスティネーション・オペランドは実効アドレス・フィールドで指定されます。

コンディション・コード：

X	N	Z	V	C
—	*	*	0	0

X－影響を受けない。
N－結果の最上位ビットがセットされれば、セット、それ以外のときはクリア。
Z－結果が0であればセット、それ以外のときはクリア。
V－常にクリア。
C－常にクリア。

命令フォーマット(ワード・フォーム)：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
										実行アドレス					
1	0	1	1	レジスタ			Opモード			モード			レジスタ		

命令フィールド：

レジスタ・フィールド——8つのデータ・レジスタのいずれかを指定。

Opモード・フィールド：

バイト	ワード	ロング・ワード	操　作
100	101	110	(< ea >) ⊕ (< Dn >) →< ea >

実効アドレス・フィールド——デスティネーション・オペランドを指定。次に示すとおり、データ可変アドレッシング・モードのみ可。

ｱﾄﾞﾚｯｼﾝｸﾞ・ﾓｰﾄﾞ	モード	レジスタ
Dn	000	reg. number : Dn
An	–	–
(An)	010	reg. number : An
(An)＋	011	reg. number : An
−(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

ｱﾄﾞﾚｯｼﾝｸﾞ・ﾓｰﾄﾞ	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	–	–
(d_{16},PC)	–	–
(d_8,PC,Xn)	–	–
(bd,PC,Xn)	–	–
([bd,PC,Xn],od)	–	–
([bd,PC],Xn,od)	–	–

注：メモリからデータ・レジスタへの操作は許されません。ほとんどのアセンブラは、ソースがイミディエイト・データのときにはEORIを使用します。

EORI

Exclusive OR Immediate・イミディエイト排他的論理和

操作：　イミディエイト・データ⊕デスティネーション→デスティネーション

アセンブラ・シンタックス：　EORI #<data>, <ea>

属性：　サイズ＝(バイト、ワード、ロング・ワード)

説明：　イミディエイト・データとデスティネーション・オペランドとの排他的論理和をとり、結果をデスティネーションに格納します。操作サイズはバイト、ワード、ロング・ワードが指定できます。イミディエイト・データのサイズは操作サイズと同じです。

コンディション・コード：

X	N	Z	V	C
—	*	*	0	0

X－影響を受けない。

N－結果の最上位ビットがセットされればセット、それ以外のときはクリア。

Z－結果が0であればセット、それ以外のときはクリア。

V－常にクリア。

C－常にクリア。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
										実効アドレス					
0	0	0	0	1	0	1	0	サイズ		モード			レジスタ		
ワード・データ（16ビット）								バイト・データ（8ビット）							
ロング・データ（32ビット）															

命令フィールド：

サイズ・フィールド――操作サイズを指定。

00－バイト操作

01－ワード操作

10－ロング・ワード操作

実効アドレス・フィールド——デスティネーション・オペランドを指定。次に示すとおり、データ可変アドレッシング・モードのみ可。

アドレッシング・モード	モード	レジスタ
Dn	000	reg. number : Dn
An	–	–
(An)	010	reg. number : An
(An)+	011	reg. number : An
–(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

アドレッシング・モード	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	–	–
(d_{16},PC)	–	–
(d_8,PC,Xn)	–	–
(bd,PC,Xn)	–	–
([bd,PC,Xn],od)	–	–
([bd,PC],Xn,od)	–	–

イミディエイト・フィールド——(命令直後のデータ)

サイズ＝00　データはイミディエイト・ワードの下位バイト。

サイズ＝01　データはイミディエイト・ワード全体。

サイズ＝10　データはその次の2つのイミディエイト・ワード。

EORI to CCR

Exclusive OR Immediate to Condition Code・
コンディション・コードに対するイミディエイト排他的論理和

操作：　ソース⊕CCR→CCR

アセンブラ・シンタックス：　EORI #<data>, CCR

属性：　サイズ＝(バイト)

説明：　イミディエイト・オペランドとコンディション・コードの排他的論理和をとり、結果をコンディション・コード・レジスタ(ステータス・レジスタの下位バイト)に格納します。コンディション・コード・レジスタの実装されている全ビットが影響を受けます。

コンディション・コード：

X	N	Z	V	C
*	*	*	*	*

X－イミディエイト・オペランドのビット4が1であれば反転、それ以外のときは変化しない。
N－イミディエイト・オペランドのビット3が1であれば反転、それ以外のときは変化しない。
Z－イミディエイト・オペランドのビット2が1であれば反転、それ以外のときは変化しない。
V－イミディエイト・オペランドのビット1が1であれば反転、それ以外のときは変化しない。
C－イミディエイト・オペランドのビット0が1であれば反転、それ以外のときは変化しない。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	0	0	1	1	1	1	0	0
0	0	0	0	0	0	0	0	バイト・データ (8ビット)							

EORI to SR

Exclusive OR Immediate to the Status Register (Privileged Instruction)・
ステータス・レジスタに対するイミディエイト排他的論理和

操作：　スーパバイザ状態ではソース⊕SR→SR
　　　　ユーザ状態ではTRAP

アセンブラ・シンタックス：　EORI #<data>, SR

属性：　サイズ＝(ワード)

説明：　イミディエイト・オペランドとステータス・レジスタの内容の排他的論理和をとり、結果をステータス・レジスタに格納します。ステータス・レジスタに実装されている全ビットが影響を受けます。

コンディション・コード：

X	N	Z	V	C
*	*	*	*	*

X－イミディエイト・オペランドのビット4が1であれば反転、それ以外のときは変化しない。
N－イミディエイト・オペランドのビット3が1であれば反転、それ以外のときは変化しない。
Z－イミディエイト・オペランドのビット2が1であれば反転、それ以外のときは変化しない。
V－イミディエイト・オペランドのビット1が1であれば反転、それ以外のときは変化しない。
C－イミディエイト・オペランドのビット0が1であれば反転、それ以外のときは変化しない。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	0	1	1	1	1	1	0	0
ワード・データ（16ビット）															

EXG

Exchange Registers・レジスタの交換

操作：　Rx⟷Ry

アセンブラ・シンタックス：　EXG Dx, Dy
EXG Ax, Ay
EXG Dx, Ay
EXG Ay, Dx

属性：　サイズ=(ロング・ワード)

説明：　2つの32ビット・レジスタ内容を交換します。命令は次の3種類の交換を実行します。

1. データ・レジスタ間の交換
2. アドレス・レジスタ間の交換
3. データ・レジスタとアドレス・レジスタ間の交換

コンディション・コード：　影響を受けない。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	レジスタRx			1	Opモード					レジスタRy		

命令フィールド：

レジスタRxフィールド――モードにより、データ・レジスタまたはアドレス・レジスタを指定。データ・レジスタとアドレス・レジスタ間の交換の場合は、このフィールドでは常にデータ・レジスタを指定する。

Opモード・フィールド――交換の種類を指定。

01000 －データ・レジスタ間
01001 －アドレス・レジスタ間
10001 －データ・アドレスとアドレス・レジスタ間

レジスタRyフィールド――モードによりデータ・レジスタまたはアドレス・レジスタを指定。データ・レジスタとアドレス・レジスタ間の交換の場合は、このフィールドは常にアドレス・レジスタを指定する。

EXT / EXTB

Sign Extend・符号拡張

操作：　デスティネーション　符号拡張→デスティネーション

アセンブラ・シンタックス：
EXT.W Dn　バイトをワードに拡張
EXT.L Dn　ワードをロング・ワードに拡張
EXTB.L Dn　バイトをロング・ワードに拡張

属性：　サイズ＝(ワード、ロング・ワード)

説明：　符号ビットを左に複写しながら、データ・レジスタのバイトをワードまたはロング・ワードに、あるいはレジスタ内のワードをワードからロング・ワードに拡張します。操作がバイトからワードへの拡張の場合、指定されたデータ・レジスタのビット［7］が、そのレジスタのビット［15：8］にコピーされます。操作がワードからロング・ワードへの拡張の場合、指定されたデータ・レジスタのビット［15］がそのレジスタのビット［31：16］にコピーされます。EXTBのフォームの場合は、指定されたデータ・レジスタのビット［7］がそのデータ・レジスタのビット［31：8］にコピーされます。

コンディション・コード：

X	N	Z	V	C
—	*	*	0	0

X－影響を受けない。
N－結果が負であればセット、それ以外のときはクリア。
Z－結果がゼロであればセット、それ以外のときはクリア。
V－常にクリア。
C－常にクリア。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	Opモード			0	0	0	レジスタ		

命令フィールド：

Opモード・フィールド――符号拡張の操作サイズを指定。
010 －データ・レジスタの下位バイトをワードに符号拡張。
011 －データ・レジスタの下位ワードをロング・ワードに符号拡張。
111 －データ・レジスタの下位バイトをロング・ワードに符号拡張。

レジスタ・フィールド――符号拡張するデータ・レジスタを指定。

ILLEGAL

Take Illegal Instruction Trap • 不当命令トラップ処理要求

操作：　SSP－2→SSP；ベクタ・オフセット→（SSP）；
SSP　－4→SSP；PC→（SSP）；
SSP　－2→SSP；SR→（SSP）；
不当命令ベクタ・アドレス→PC

アセンブラ・シンタックス：　ILLEGAL

属性：　サイズなし

説明：　ベクタ番号4の不当命令例外を発生します。他の不当命令ビット・パターンはすべて、将来の命令セットの拡張のために予約されています。

コンディション・コード：　影響を受けない。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	0	1	1	1	1	1	1	0	0

JMP

Jump・ジャンプ

操作：　デスティネーション・アドレス→PC
アセンブラ・シンタックス：　JMP < ea >
属性：　サイズなし
説明：　プログラムの実行は、命令で指定されるアドレスから続行されます。実効アドレスのアドレッシング・モードは、制御アドレッシング・モードでなければなりません。

コンディション・コード：　影響を受けない。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	1	1	実効アドレス モード			実効アドレス レジスタ		

命令フィールド：

実効アドレス・フィールド――次の命令のアドレスを指定。次に示すとおり、制御アドレッシング・モードのみ可。

ｱﾄﾞﾚｯｼﾝｸﾞ･ﾓｰﾄﾞ	モード	レジスタ
Dn	−	−
An	−	−
(An)	010	reg. number : An
(An)＋	−	−
−(An)	−	−
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

ｱﾄﾞﾚｯｼﾝｸﾞ･ﾓｰﾄﾞ	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	−	−
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

JSR

Jump to Subroutine • サブルーチンへのジャンプ

操作： SP－4→SP；PC→（SP）
デスティネーション・アドレス→PC

アセンブラ・シンタックス： JSR <ea>

属性： サイズなし

説明： JSR命令の直後の命令のロング・ワード・アドレスをシステム・スタックにプッシュします。プログラムの実行は命令で指定したアドレスから続行されます。

コンディション・コード： 影響を受けない。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
										実効アドレス					
0	1	0	0	1	1	1	0	1	0	モード			レジスタ		

命令フィールド：

実効アドレス・フィールド――次の命令のアドレスを指定。次に示すとおり、制御アドレッシング・モードのみ可。

アドレッシング・モード	モード	レジスタ
Dn	–	–
An	–	–
(An)	010	reg. number : An
(An)＋	–	–
－(An)	–	–
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

アドレッシング・モード	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	–	–
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

LEA

Load Effective Address • 実効アドレスのロード

操作：　< ea >→An

アセンブラ・シンタックス：　LEA < ea >, An

属性：　サイズ＝(ロング・ワード)

説明：　指定したアドレス・レジスタに実効アドレスをロードします。アドレス・レジスタの32ビット全体がこの命令の影響を受けます。

コンディション・コード：　影響を受けない。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	レジスタ			1	1	1	実効アドレス モード			実効アドレス レジスタ		

命令フィールド：

レジスタ・フィールド――実効アドレスで更新されるアドレス・レジスタを指定。

実効アドレス・フィールド――アドレス・レジスタにロードするアドレスを指定。次に示すとおり、制御アドレッシング・モードのみ可。

ｱﾄﾞﾚｯｼﾝｸﾞ･ﾓｰﾄﾞ	モード	レジスタ
Dn	−	−
An	−	−
(An)	010	reg. number : An
(An)＋	−	−
−(An)	−	−
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

ｱﾄﾞﾚｯｼﾝｸﾞ･ﾓｰﾄﾞ	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	−	−
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

LINK

Link and Allocate・リンクと割付け

操作：　SP－4→Sp；An→（SP）；
SP→An；SP＋d→SP

アセンブラ・シンタックス：　LINK　An, ＃<displacement>

属性：　サイズ＝(ワード、ロング・ワード)

説明：　指定されたアドレス・レジスタの現在の内容をスタックにプッシュします。その後、そのアドレス・レジスタに更新されたスタック・ポインタの値をロードします。最後にディスプレースメントをスタック・ポインタに加えます。ワード・サイズの操作の場合、ディスプレースメントはオペレーション・ワードの次の符号拡張されたワードです。ロング・サイズ操作の場合、ディスプレースメントはオペレーション・ワードの次のロング・ワードです。アドレス・レジスタはスタック内で1ロング・ワードを占有します。ユーザはスタック領域を割り付けるために、負のディスプレースメントを指定しなければなりません。

コンディション・コード：　影響を受けない。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	1	0	レジスタ		
ワード・ディスプレースメント															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	0	0	0	1	レジスタ		
上位ディスプレースメント															
下位ディスプレースメント															

命令フィールド：

レジスタ・フィールド――リンクのアドレス・レジスタを指定。

ディスプレースメント・フィールド――スタック・ポインタに加算する2の補数の整数を指定。

注：LINKとUNLKを使用すれば、ネストしたサブルーチン・コールのスタックのローカル・データとパラメータ領域のリンク・リストを維持できます。

LSL / LSR

Logical Shift・論理シフト

操作：　デスティネーション　Shifted by <カウント>→デスティネーション

アセンブラ・シンタックス：　LSd Dx, Dy

LSd #　< data >, Dy

LSd　< ea >

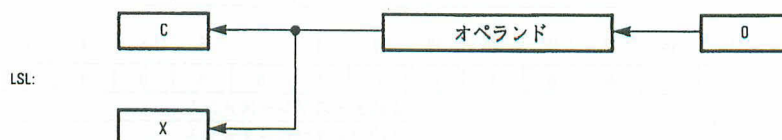
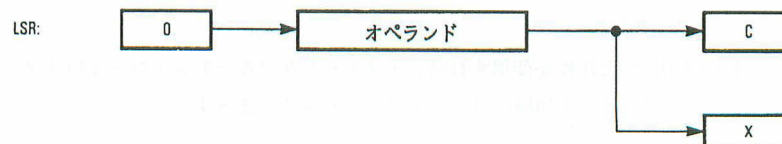
ここで、dは方向、L（左）またはR（右）。

属性：　サイズ＝(バイト、ワード、ロング・ワード)

説明：　オペランドのビットを指定方向(左または右)にシフトします。キャリ・ビットは、オペランドから最後に送り出されたビットを受け取ります。レジスタをシフトするためのシフト・カウントは次の2とおりの方法で指定できます。

1. イミディエイト――シフト・カウント(1-8)は命令の中で指定します。
2. レジスタ――シフト・カウント(モジュロ64)は命令で指定するデータ・レジスタにあります。

レジスタ・デスティネーションの操作サイズは、バイト、ワード、ロング・ワードが指定できます。ただし、メモリ内容、< ea >、は1ビットしかシフトできず、オペランド・サイズもワードに限定されます。

LSL命令は、オペランドをシフト・カウントで指定された位置だけ左にシフトします。最上位ビットから送り出されたビットはキャリ、拡張の両ビットに入ります。最下位ビットには0がシフトされます。

LSR命令は、オペランドをシフト・カウントで指定された位置だけ右にシフトします。最下位ビットから送り出されたビットはキャリ、拡張の両ビットに入ります。最上位ビットには0がシフトされます。

コンディション・コード：

X	N	Z	V	C
*	*	*	0	*

X－オペランドから最後に送り出されたビットに従ってセット、シフト・カウントが0のときは影響されない。
N－結果が負であればセット、それ以外のときはクリア。
Z－結果が0であればセット、それ以外のときはクリア。
V－常にクリア。
C－オペランドから最後に送り出されたビットに従ってセット、シフト・カウントが0のときはクリア。

命令フォーマット(レジスタ・シフト)：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	カウント/レジスタ			dr	サイズ		i/r	0	1	レジスタ		

命令フィールド(レジスタ・シフト)：

カウント/レジスタ・フィールド：
i/r＝0　このフィールドでシフト・カウントを指定。1-7の値は1-7、0は8を表わす。
i/r＝1　シフト・カウント(モジュロ64)はこのフィールドで指定するデータ・レジスタにある。

drフィールド——シフト方向を指定。
0－右シフト
1－左シフト

サイズ・フィールド——操作サイズを指定。
00－バイト操作
01－ワード操作
10－ロング・ワード操作

i/rフィールド：
i/r＝0　シフト・カウントをイミディエイト値で指定。
i/r＝1　シフト・カウントをデータ・レジスタで指定。

レジスタ・フィールド——シフトするデータ・レジスタを指定。

命令フォーマット(メモリ・シフト)：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
										実効アドレス					
1	1	1	0	0	0	1	dr	1	1	モード			レジスタ		

命令フィールド(メモリ・シフト)：

dr フィールド――シフト方向を指定。

0－右シフト

1－左シフト

実効アドレス・フィールド――シフトするオペランドを指定。次に示すとおり、メモリ可変アドレッシング・モードのみ可。

アドレッシング・モード	モード	レジスタ
Dn	－	－
An	－	－
(An)	010	reg. number : An
(An)＋	011	reg. number : An
－(An)	100	reg. number : An
$(d_{16}$,An)	101	reg. number : An
$(d_8$,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

アドレッシング・モード	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	－	－
$(d_{16}$,PC)	－	－
$(d_8$,PC,Xn)	－	－
(bd,PC,Xn)	－	－
([bd,PC,Xn],od)	－	－
([bd,PC],Xn,od)	－	－

MOVE

Move Data from Source to Destination •
ソースからデスティネーションへのデータ転送

操作：　ソース→デスティネーション

アセンブラ・シンタックス：　MOVE ＜ea＞, ＜ea＞

属性：　サイズ＝(バイト、ワード、ロング・ワード)

説明：　ソースのデータをデスティネーション・ロケーションに転送し、データの内容に従ってコンディション・コードをセットします。操作サイズはバイト、ワード、ロング・ワードが指定できます。

コンディション・コード：

X	N	Z	V	C
—	*	*	0	0

X－影響を受けない。

N－結果が負であればセット、それ以外のときはクリア。

Z－結果が0であればセット、それ以外のときはクリア。

V－常にクリア。

C－常にクリア。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	サイズ		デスティネーション						ソース					
				レジスタ			モード			モード			レジスタ		

命令フィールド：

サイズ・フィールド——転送するオペランドのサイズを指定。

01－バイト操作

11－ワード操作

10－ロング・ワード操作

デスティネーション実効アドレス・フィールド——デスティネーション・ロケーションを指定。次に示すとおり、データ可変アドレッシング・モードのみ可。

アドレッシング・モード	モード	レジスタ
Dn	000	reg. number : Dn
An	–	–
(An)	010	reg. number : An
(An)+	011	reg. number : An
–(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

アドレッシング・モード	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#<data>	–	–
(d_{16},PC)	–	–
(d_8,PC,Xn)	–	–
(bd,PC,Xn)	–	–
([bd,PC,Xn],od)	–	–
([bd,PC],Xn,od)	–	–

ソース実効アドレス・フィールド——ソース・オペランドを指定。次に示すとおり、すべてのアドレッシング・モードが可。

アドレッシング・モード	モード	レジスタ
Dn	000	reg. number : Dn
An*	001	reg. number : An
(An)	010	reg. number : An
(An)+	011	reg. number : An
–(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

アドレッシング・モード	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

* 操作サイズがバイトの場合、アドレス・レジスタ直接は不可。

注：1. デスティネーションがアドレス・レジスタのときには、ほとんどのアセンブラがMOVEAを使用します。

2. イミディエイト8ビット値をデータ・レジスタに転送するときには、MOVEQを使用することができます。

MOVEA

Move Address・アドレス転送

操作：　ソース→デスティネーション

アセンブラ・シンタックス：　MOVEA　<ea>，An

属性：　サイズ＝(ワード、ロング・ワード)

説明：　ソース・オペランドの内容をデスティネーションのアドレス・レジスタに転送します。操作サイズはワードまたはロング・ワードが指定できます。ワード・サイズのソース・オペランドは32ビットに符号拡張されます。

コンディション・コード：　影響を受けない。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	サイズ		デスティネーション レジスタ			0	0	1	ソース モード			ソース レジスタ		

命令フィールド：

サイズ・フィールド――転送するオペランドのサイズを指定。

11－ワード操作。ただし、ソース・オペランドはロング・オペランドに符号拡張され、32ビット全体がアドレス・レジスタにロードされる。

10－ロング操作

デスティネーション・レジスタ・フィールド――デスティネーションのアドレス・レジスタを指定。

実効アドレス・フィールド――ソース・オペランドのロケーションを指定。次に示すとおり、すべてのアドレッシング・モードが可。

ｱﾄﾞﾚｯｼﾝｸﾞ・ﾓｰﾄﾞ	モード	レジスタ
Dn	000	reg. number : Dn
An	001	reg. number : An
(An)	010	reg. number : An
(An)＋	011	reg. number : An
－(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

ｱﾄﾞﾚｯｼﾝｸﾞ・ﾓｰﾄﾞ	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

MOVE from CCR

Move from the Condition Code Register・
コンディション・コード・レジスタからの転送

操作：　CCR→デスティネーション

アセンブラ・シンタックス：　MOVE CCR, < ea >

属性：　サイズ＝(ワード)

説明：　コンディション・コード・ビット(ワード・サイズにゼロ拡張されたもの)をデスティネーション・ロケーションに転送します。オペランドのサイズはワードです。実装されていないビットはゼロで読み出されます。

コンディション・コード：　影響を受けない。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	1	0	1	1	実効アドレス					
										モード			レジスタ		

命令フィールド：

実効アドレス・フィールド――デスティネーション・ロケーションを指定。次に示すとおり、データ可変アドレッシング・モードのみ可。

アドレッシング・モード	モード	レジスタ
Dn	000	reg. number : Dn
An	–	–
(An)	010	reg. number : An
(An)＋	011	reg. number : An
－(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

アドレッシング・モード	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	–	–
(d_{16},PC)	–	–
(d_8,PC,Xn)	–	–
(bd,PC,Xn)	–	–
([bd,PC,Xn],od)	–	–
([bd,PC],Xn,od)	–	–

注：MOVE from CCR はワード操作で、ANDI、ORI およびEORI to CCR はバイト操作です。

MOVE to CCR

Move to Condition Codes **・コンディション・コードへの転送**

操作：　ソース→CCR

アセンブラ・シンタックス：　MOVE < ea >, CCR

属性：　サイズ＝(ワード)

説明：　ソース・オペランドの下位バイトをコンディション・コード・レジスタに転送します。ソース・オペランドの上位バイトは無視されます。ステータス・レジスタの上位バイトは変更されません。

コンディション・コード：

X	N	Z	V	C
*	*	*	*	*

X－ソース・オペランドのビット4と同じ値にセットされる。
N－ソース・オペランドのビット3と同じ値にセットされる。
Z－ソース・オペランドのビット2と同じ値にセットされる。
V－ソース・オペランドのビット1と同じ値にセットされる。
C－ソース・オペランドのビット0と同じ値にセットされる。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	0	0	1	1	実効アドレス モード			 レジスタ		

命令フィールド：

実効アドレス・フィールド――ソース・オペランドを指定。次に示すとおり、データ・アドレッシング・モードのみ可。

ｱﾄﾞﾚｯｼﾝｸﾞ・ﾓｰﾄﾞ	モード	レジスタ
Dn	000	reg. number : Dn
An	－	－
(An)	010	reg. number : An
(An)＋	011	reg. number : An
－(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

ｱﾄﾞﾚｯｼﾝｸﾞ・ﾓｰﾄﾞ	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

注：MOVE to CCR はワード操作で、ANDI、ORI およびEORI to CCR はバイト操作です。

MOVE from SR

Move from the Status Register(Privileged Instruction)・
ステータス・レジスタからの転送(特権命令)

操作：　スーパバイザ状態ではSR→デスティネーション
ユーザ状態ではTRAP

アセンブラ・シンタックス：　MOVE　SR, < ea >

属性：　サイズ＝(ワード)

説明：　ステータス・レジスタ内のデータをデスティネーション・ロケーションに転送します。デスティネーションはワード長です。実装されていないビットはゼロで読み出されます。

コンディション・コード：　影響を受けない。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	1	1	実効アドレス					
										モード			レジスタ		

命令フィールド：

実効アドレス・フィールド――デスティネーション・ロケーションを指定。次に示すとおり、データ可変アドレッシング・モードのみ可。

ｱﾄﾞﾚｯｼﾝｸﾞ・ﾓｰﾄﾞ	モード	レジスタ
Dn	000	reg. number : Dn
An	−	−
(An)	010	reg. number : An
(An)＋	011	reg. number : An
−(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

ｱﾄﾞﾚｯｼﾝｸﾞ・ﾓｰﾄﾞ	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	−	−
(d_{16},PC)	−	−
(d_8,PC,Xn)	−	−
(bd,PC,Xn)	−	−
([bd,PC,Xn],od)	−	−
([bd,PC],Xn,od)	−	−

注：コンディション・コードだけにアクセスするには、MOVE　from　CCR命令を使用します。

MOVE to SR

Move to the Status Register(Privileged Instruction)

ステータス・レジスタへの転送(特権命令)

操作：　スーパバイザ状態ではソース→SR
ユーザ状態ではTRAP

アセンブラ・シンタックス：　MOVE <ea>, SR

属性：　サイズ=(ワード)

説明：　ソース・オペランド内のデータをステータス・レジスタに転送します。ソース・オペランドはワードで、ステータス・レジスタに実装されている全ビットが影響を受けます。

コンディション・コード：　ソース・オペランドに従ってセットされます。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
										実効アドレス					
										モード			レジスタ		
0	1	0	0	0	1	1	0	1	1						

命令フィールド：

実効アドレス・フィールド——ソース・オペランドを指定。次に示すとおり、データ・アドレッシング・モードのみ可。

アドレッシング・モード	モード	レジスタ
Dn	000	reg. number : Dn
An	–	–
(An)	010	reg. number : An
(An)+	011	reg. number : An
–(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

アドレッシング・モード	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

MOVE USP

Move User Stack Pointer(Privileged Instruction)・

ユーザ・スタック・ポインタの転送(特権命令)

操作：　スーパバイザ状態ではUSP→AnまたはAn→USP

ユーザ状態ではTRAP

アセンブラ・シンタックス：　MOVE USP, An

MOVE An, USP

属性：　サイズ＝(ロング・ワード)

説明：　ユーザ・スタック・ポインタの内容を指定されたアドレス・レジスタへ、あるいは指定されたアドレス・レジスタの内容をユーザ・スタック・ポインタへ転送します。

コンディション・コード：　影響を受けない。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	0	dr	レジスタ		

命令フィールド：

drフィールド——転送の方向を指定。

0－アドレス・レジスタをユーザ・スタック・ポインタへ転送。

1－ユーザ・スタック・ポインタをアドレス・レジスタへ転送。

レジスタ・フィールド——操作の対象となるアドレス・レジスタを指定。

MOVEC

Move Control Register(Privileged Instruction)・制御レジスタ転送(特権命令)

操作：　スーパバイザ状態ではRc→Rn またはRn→Rc
ユーザ状態ではTRAP

アンセブラ・シンタックス：　MOVEC　Rc, Rn
MOVEC　Rn, Rc

属性：　サイズ＝(ロング・ワード)

説明：　指定された制御レジスタ(Rc)の内容を指定されたた汎用レジスタに、あるいは指定された汎用レジスタの内容を指定された制御レジスタにコピーします。この操作は制御レジスタに実装されているビット数に関係なく、常に32ビットで転送を行ないます。実装されていないビットは0で読み出されます。

コンディション・コード：　影響を受けない。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	1	0	1	dr
A/D	レジスタ			制御レジスタ											

命令フィールド：

drフィールド――転送の方向を指定。
0－制御レジスタから汎用レジスタ。
1－汎用レジスタから制御レジスタ。

A/Dフィールド――汎用レジスタの種類を指定。
0－データ・レジスタ
1－アドレス・レジスタ

レジスタ・フィールド――レジスタ番号を指定。

制御レジスタ・フィールド――制御レジスタを指定。

16進	レジスタ名/機能
000	ソース・ファンクション・コード(SFC)レジスタ
001	デスティネーション・ファンクション・コード(DFC)レジスタ
002	キャッシュ制御レジスタ(CACR)
800	ユーザ・スタック・ポインタ(USP)
801	ベクタ・ベース・レジスタ(VBR)
802	キャッシュ・アドレス・レジスタ(CAAR)
803	マスタ・スタック・ポインタ(MSP)
804	割込みスタック・ポインタ(ISP)

その他のコードはすべて不当命令例外を発生する。

MOVEM

Move Multiple Registers • 複数レジスタ転送

操作：　レジスタ→デスティネーション
ソース→レジスタ

アセンブラ・シンタックス：　MOVEM register list, < ea >
MOVEM < ea >, register list

属性：　サイズ＝(ワード、ロング・ワード)

説明：　選択されたレジスタ群と、実効アドレスで指定されるアドレスから始まる連続したメモリ・ロケーションとの間でデータ転送を行ないます。マスク・フィールドの対応するビットが1にセットされているレジスタが選択されます。命令サイズで各レジスタの16ビットを転送するか、32ビット全体を転送するかを決めます。ワードをアドレス・レジスタまたはデータ・レジスタに転送する場合、各ワードは32ビットに符号拡張され、ロング・ワードとしてレジスタにロードされます。

アドレッシング・モードを選択すれば、MOVEM命令の動作モードも選択され、制御モード、プリデクリメント・モードおよびポストインクリメント・モードの3つだけが有効です。実効アドレスが制御モードの1つで指定された場合、レジスタ群は指定されたアドレスから転送され、アドレスは1回の転送が終わるたびにオペランド長(2または4)ずつ増加していきます。転送はデータ・レジスタ0からデータ・レジスタ7、ついでアドレス・レジスタ0からアドレス・レジスタ7の順に行なわれます。

実効アドレスがプリデクリメント・モードで指定された場合は、レジスタからメモリへの操作だけが可能です。レジスタは指定されたアドレス－オペランド長(2または4)から格納され、アドレスは1回の転送が終わるたびにオペランド長ずつ減少していきます。格納はアドレス・レジスタ7からアドレス・レジスタ0、ついでデータ・レジスタ7からデータ・レジスタ0の順に行なわれます。命令が完了すると、デクリメントされたアドレス・レジスタは、最後に格納したオペランドのアドレスを保持しています。MC68030では、アドレッシング・レジスタもメモリに転送される場合、デクリメントされた値が書き込まれます。

実効アドレスがポストインクリメント・モードで指定される場合は、メモリからレジスタへの転送のみ可能です。レジスタは指定されたアドレスからロードされ、アドレスは1回の転送が終わるたびにオペランド長(2または4)ずつインクリメントされていきます。ロードされる順序は、制御モード・アドレッシングの場合と同じです。命令完了後には、インクリメントされたアドレス・レジスタは、最後にロードされたオペランドのアドレス＋オペランド長(2または4)を保持しています。MC68030では、アドレッシング・レジスタもメモリからロードされる場合は、ロードされる値はフェッチされた値＋オペランド長になります。

コンディション・コード：　影響を受けない。

命令フォーマット：

<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td rowspan="2">0</td><td rowspan="2">1</td><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">1</td><td rowspan="2">dr</td><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">1</td><td rowspan="2">サイズ</td><td colspan="6">実効アドレス</td></tr>
<tr><td colspan="3">モード</td><td colspan="3">レジスタ</td></tr>
<tr><td colspan="16">レジスタ・リスト・マスク</td></tr>
</table>

命令フィールド：

drフィールド——転送の方向を指定。

0－レジスタからメモリ

1－メモリからレジスタ

サイズ・フィールド——転送するレジスタのサイズを指定。

0－ワード転送

1－ロング・ワード転送

実効アドレス・フィールド——メモリ・アドレスを指定。レジスタからメモリへの転送の場合は、次に示すとおり、制御可変アドレッシング・モードまたはプリデクリメント・アドレッシング・モードのみ可。

ｱﾄﾞﾚｯｼﾝｸﾞ･ﾓｰﾄﾞ	モード	レジスタ
Dn	－	－
An	－	－
(An)	010	reg. number : An
(An)＋	－	－
－(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

ｱﾄﾞﾚｯｼﾝｸﾞ･ﾓｰﾄﾞ	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	－	－
(d_{16},PC)	－	－
(d_8,PC,Xn)	－	－
(bd,PC,Xn)	－	－
([bd,PC,Xn],od)	－	－
([bd,PC],Xn,od)	－	－

メモリからレジスタへの転送の場合は、次に示すとおり、制御アドレッシング・モードまたはポストインクリメント・アドレッシング・モードのみ可。

ｱﾄﾞﾚｯｼﾝｸﾞ･ﾓｰﾄﾞ	モード	レジスタ
Dn	－	－
An	－	－
(An)	010	reg. number : An
(An)＋	011	reg. number : An
－(An)	－	－
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

ｱﾄﾞﾚｯｼﾝｸﾞ･ﾓｰﾄﾞ	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	－	－
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

レジスタ・リスト・マスク・フィールド――転送するレジスタを指定。下位ビットは最初に転送するレジスタに対応し、上位ビットは最後に転送するレジスタに対応する。制御モードおよびポストインクリメント・アドレッシング・モードの場合、マスクとレジスタの対応は次のとおり。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A7	A6	A5	A4	A3	A2	A1	A0	D7	D6	D5	D4	D3	D2	D1	D0

プリデクリメント・アドレッシング・モードの場合、マスクとレジスタの対応は次のとおり。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D0	D1	D2	D3	D4	D5	D6	D7	A0	A1	A2	A3	A4	A5	A6	A7

注：メモリ・オペランドのリード・バス・サイクルが1サイクル余分に発生します。このサイクルでは必要な最後のレジスタ・イメージより1つ上位のアドレスにあるオペランドがアクセスされます。

MOVEP

Move Peripheral Data • 周辺データ転送

操作：　ソース→デスティネーション

アセンブラ・シンタックス：　MOVEP Dx, (d, Ay)
MOVEP (d, Ay), Dx

属性：　サイズ＝(ワード、ロング・ワード)

説明：　データ・レジスタと指定したロケーションから開始し2ずつアドレスが増加するアドレス空間内のオルタネート・バイトとの間でデータ転送を行ないます。この命令は8ビットの周辺デバイスを16ビットのデータ・バスに接続するためのものです。データ・レジスタの上位バイトが最初に転送され、下位バイトが最後に転送されます。メモリ・アドレスはアドレス・レジスタ間接＋16ビット・ディスプレースメント・アドレッシング・モードで指定されます。アドレスが偶数の場合、転送はすべてデータ・バスの上位半分で行なわれ、奇数の場合はすべてデータ・バスの下位半分で行なわれます。この命令は8ビットまたは32ビットのバス上でも、1バイトおきにアクセスを行ないます。

例：偶数アドレスとの間のロング・ワード転送

レジスタのバイト構成

31　　24	23　　16	15　　8	7　　0
上位	中上位	中下位	下位

メモリのバイト構成(先頭が下位アドレス)

15　　8	7　　0
上位	
中上位	
中下位	
下位	

例：奇数アドレスとの間のワード転送

レジスタのバイト構成

31　　24	23　　16	15　　8	7　　0
		上位	下位

メモリのバイト構成(先頭が下位アドレス)

15　　8	7　　0
上位	
下位	

コンディション・コード： 影響を受けない。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	データ・レジスタ			Opモード			0	0	1	アドレス・レジスタ		
ディスプレースメント（16ビット）															

命令フィールド：

データ・レジスタ・フィールド──データ転送の対象となるデータ・レジスタを指定。

Opモード・フィールド──操作サイズと方向を指定。

100 －メモリからレジスタへのワード転送
101 －メモリからレジスタへのロング・ワード転送
110 －レジスタからメモリへのワード転送
111 －レジスタからメモリへのロング・ワード転送

アドレス・レジスタ・フィールド──ディスプレースメント付きアドレス・レジスタ間接アドレッシング・モードで使用するアドレス・レジスタを指定。

ディスプレースメント・フィールド──オペランド・アドレスに使用するディスプレースメントを指定。

MOVEQ

Move Quick **・クイック転送**

操作：　イミディエイト・データ→デスティネーション

アセンブラ・シンタックス：　MOVEQ　#< data >, Dn

属性：　サイズ＝(ロング・ワード)

説明：　イミディエイト・データの1バイトを32ビットのデータ・レジスタに転送します。データはオペレーション・ワードの8ビット・フィールドにあり、ロング・ワードに符号拡張されてデータ・レジスタに転送されます。

コンディション・コード：

X	N	Z	V	C
—	*	*	0	0

X－影響を受けない。

N－結果が負であればセット、それ以外のときはクリア。

Z－結果が0であればセット、それ以外のときはクリア。

V－常にクリア。

C－常にクリア。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	レジスタ			0	データ							

命令フィールド：

レジスタ・フィールド——ロードするデータ・レジスタを指定。

データ・フィールド——ロング・ワードに符号拡張する8ビット・データ。

MOVES

Move Address Space(Privileged Instruction) • アドレス空間転送(特権命令)

操作：　スーパバイザ状態ではRn→デスティネーション［DFC］
またはソース［SFC］→Rn
ユーザ状態ではTRAP

アセンブラ・シンタックス：　MOVES Rn, < ea >
MOVES < ea >, Rn

属性：　サイズ＝(バイト、ワード、ロング・ワード)

説明：　指定された汎用レジスタからデスティネーション・ファンクション・コード(DFC)レジスタで指定されるアドレス空間内のロケーションへ、バイト、ワードまたはロング・ワード・オペランドを転送します。あるいは、ソース・ファンクション・コード(SFC)レジスタで指定されるアドレス空間内のロケーションから指定された汎用レジスタへ、バイト、ワードまたはロング・ワード・オペランドを転送します。

デスティネーションがデータ・レジスタの場合、ソース・オペランドは操作サイズに応じてそのデータ・レジスタに対応する下位ビットを交換します。デスティネーションがアドレス・レジスタの場合、ソース・オペランドは32ビットに符号拡張されて、そのアドレス・レジスタにロードされます。

コンディション・コード：　影響を受けない。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	1	0	サイズ		実効アドレス					
										モード			レジスタ		
A/D	レジスタ			dr	0	0	0	0	0	0	0	0	0	0	0

命令フィールド：

サイズ・フィールド――操作サイズを指定。
00－バイト操作
01－ワード操作
10－ロング・ワード操作

実効アドレス・フィールド――オルタネート・アドレス空間内のソースまたはデスティネーション・ロケーションを指定。次に示すとおり、メモリ可変アドレッシング・モードのみ可。

アドレッシング・モード	モード	レジスタ
Dn	−	−
An	−	−
(An)	010	reg. number : An
(An)＋	011	reg. number : An
−(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

アドレッシング・モード	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	−	−
(d_{16},PC)	−	−
(d_8,PC,Xn)	−	−
(bd,PC,Xn)	−	−
([bd,PC,Xn],od)	−	−
([bd,PC],Xn,od)	−	−

A/Dフィールド――汎用レジスタの種類を指定。
0－データ・レジスタ
1－アドレス・レジスタ
レジスタ・フィールド――レジスタ番号を指定。
drフィールド――転送方向を指定。
0－＜ea＞から汎用レジスタ
1－汎用レジスタから＜ea＞

注：次の2つの例は、ソースおよびデスティネーションに対して同じアドレス・レジスタを使用しているため、格納される値は不定となります。

MOVES.x　An,　(An)＋
MOVES.x　An,　－(An)

MC68010、MC68020およびMC68030 の現在のインプリメンテーションは、Anをインクリメントまたはデクリメントした値を格納します。

MULS

Signed Multiply・符号付き乗算

操作：　ソース＊デスティネーション→デスティネーション

アセンブラ・シンタックス：
MULS.W <ea>, Dn　16×16→32
MULS.L <ea>, Dl　32×32→32
MULS.L <ea>, Dh：Dl　32×32→64

属性：　サイズ＝(ワード、ロング・ワード)

説明：　2つの符号付きオペランドを乗算して符号付きの結果を生成します。この命令にはワード・オペランド・フォームとロング・ワード・オペランド・フォームがあります。ワード・フォームでは、乗数と被乗数が両方ともワード・オペランドで、結果はロング・ワード・オペランドになります。レジスタ・オペランドはその下位ワードで、上位ワードは無視されます。全32ビットの積がデスティネーション・データ・レジスタにセーブされます。

ロング・フォームでは、乗数と被乗数がともにロング・ワード・オペランドで、結果はロング・ワードまたはクワッド・ワードのいずれかになります。ロング・ワードの結果は、クワッド・ワードの結果の下位32ビットです。積の上位32ビットは捨てられます。

コンディション・コード：

X	N	Z	V	C
—	*	*	*	0

X－影響を受けない。

N－結果が負であればセット、それ以外のときはクリア。

Z－結果が0であればセット、それ以外のときはクリア。

V－オーバフローがあればセット、それ以外のときはクリア。

C－常にクリア。

注：オーバフロー(V＝1)は、32ビットのオペランドを乗算して32ビットの結果を生成する場合にのみ発生する可能性があります。クワッド・ワード積の上位32ビットが下位32ビットの符号拡張でない場合には、オーバフローが発生します。

命令フォーマット(ワード・フォーム)：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	レジスタ			1	1	1	実効アドレス モード			実効アドレス レジスタ		

命令フィールド：

レジスタ・フィールド――データ・レジスタをデスティネーションとして指定。

実効アドレス・フィールド――ソース・オペランドを指定。次に示すとおり、データ・アドレッシング・モードのみ可。

アドレッシング・モード	モード	レジスタ	アドレッシング・モード	モード	レジスタ
Dn	000	reg. number : Dn	(xxx).W	111	000
An	–	–	(xxx).L	111	001
(An)	010	reg. number : An	#〈data〉	111	100
(An)+	011	reg. number : An			
–(An)	100	reg. number : An			
(d_{16},An)	101	reg. number : An	(d_{16},PC)	111	010
(d_8,An,Xn)	110	reg. number : An	(d_8,PC,Xn)	111	011
(bd,An,Xn)	110	reg. number : An	(bd,PC,Xn)	111	011
([bd,An,Xn],od)	110	reg. number : An	([bd,PC,Xn],od)	111	011
([bd,An],Xn,od)	110	reg. number : An	([bd,PC],Xn,od)	111	011

命令フォーマット（ロング・フォーム）：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	0	0	実効アドレス モード			実効アドレス レジスタ		
0	レジスタDq			1	サイズ	0	0	0	0	0	0	0	レジスタDr		

命令フィールド：

実効アドレス・フィールド――ソース・オペランドを指定。次に示すとおり、データ・アドレッシング・モードのみ可。

アドレッシング・モード	モード	レジスタ	アドレッシング・モード	モード	レジスタ
Dn	000	reg. number : Dn	(xxx).W	111	000
An	–	–	(xxx).L	111	001
(An)	010	reg. number : An	#〈data〉	111	100
(An)+	011	reg. number : An			
–(An)	100	reg. number : An			
(d_{16},An)	101	reg. number : An	(d_{16},PC)	111	010
(d_8,An,Xn)	110	reg. number : An	(d_8,PC,Xn)	111	011
(bd,An,Xn)	110	reg. number : An	(bd,PC,Xn)	111	011
([bd,An,Xn],od)	110	reg. number : An	([bd,PC,Xn],od)	111	011
([bd,An],Xn,od)	110	reg. number : An	([bd,PC],Xn,od)	111	011

レジスタDlフィールド――デスティネーション・オペランドのデータ・レジスタを指定。32ビットの被乗数はこのレジスタから得られ、積の下位32ビットがこのレジスタにロードされる。

サイズ・フィールド――32ビットまたは64ビットの積を指定。

0－32ビットの積がレジスタDlに返される。

1－64ビットの積がレジスタDh：Dlに返される。

レジスタDhフィールド――サイズが1であれば、積の上位32ビットをロードするデータ・レジスタを指定。Dh＝Dlで、サイズが1であれば、演算結果は不定となる。

上記以外のときには、このフィールドは使用されない。

MULU

Unsigned Multiply・符号なし乗算

操作：　ソース＊デスティネーション→デスティネーション

アセンブラ・シンタックス：

MULU.W <ea>, Dn	16×16→32
MULU.L <ea>, Dl	32×32→32
MULU.L <ea>, Dh：DL	32×32→64

属性：　サイズ＝(ワード、ロング・ワード)

説明：　2つの符号なしオペランドを乗算して符号なしの結果を生成します。この命令にはワード・オペランド・フォームとロング・ワード・オペランド・フォームがあります。

ワード・フォームでは、乗数と被乗数が両方ともワード・オペランドで、結果はロング・ワード・オペランドになります。レジスタ・オペランドはその下位ワードで、上位ワードは無視されます。全32ビットの積がデスティネーション・データ・レジスタにセーブされます。

ロング・フォームでは、乗数と被乗数がともにロング・ワード・オペランドで、結果はロング・ワードまたはクワッド・ワードのいずれかになります。ロング・ワードの結果は、クワッド・ワードの結果の下位32ビットです。積の上位32ビットは捨てられます。

コンディション・コード：

X	N	Z	V	C
—	*	*	*	0

X－影響を受けない。
N－結果が負であればセット、それ以外のときはクリア。
Z－結果が0であればセット、それ以外のときはクリア。
V－オーバフローがあればセット、それ以外のときはクリア。
C－常にクリア。

注：オーバフロー(V＝1)は、32ビットのオペランドを乗算して32ビットの結果を生成する場合にのみ発生する可能性があります。クワッド・ワード積の上位32ビットがゼロ以外の場合には、オーバフローが発生します。

命令フォーマット(ワード・フォーム)：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	レジスタ			0	1	1	実効アドレス					
										モード			レジスタ		

命令フィールド：

レジスタ・フィールド――データ・レジスタをデスティネーションとして指定。

実効アドレス・フィールド――ソース・オペランドを指定。次に示すとおり、データ・アドレッシング・モードのみ可。

ｱﾄﾞﾚｯｼﾝｸﾞ･ﾓｰﾄﾞ	モード	レジスタ
Dn	000	reg. number : Dn
An	–	–
(An)	010	reg. number : An
(An)+	011	reg. number : An
−(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

ｱﾄﾞﾚｯｼﾝｸﾞ･ﾓｰﾄﾞ	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

命令フォーマット（ロング・フォーム）：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	0	0	実効アドレス モード			実効アドレス レジスタ		
0	レジスタDl			0	サイズ	0	0	0	0	0	0	0	レジスタDh		

命令フィールド：

実効アドレス・フィールド――ソース・オペランドを指定。次に示すとおり、データ・アドレッシング・モードのみ可。

ｱﾄﾞﾚｯｼﾝｸﾞ･ﾓｰﾄﾞ	モード	レジスタ
Dn	000	reg. number : Dn
An	–	–
(An)	010	reg. number : An
(An)+	011	reg. number : An
−(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

ｱﾄﾞﾚｯｼﾝｸﾞ･ﾓｰﾄﾞ	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

レジスタDlフィールド――デスティネーション・オペランドのデータ・レジスタを指定。32ビットの被乗数はこのレジスタから得られ、積の下位32ビットがこのレジスタにロードされる。
サイズ・フィールド――32ビットまたは64ビットの積を指定。
　0－32ビットの積がレジスタDlに返される。
　1－64ビットの積がレジスタDh：Dlに返される。
レジスタDhフィールド――サイズが1であれば、積の上位32ビットをロードするデータ・レジスタを指定。Dh＝Dlで、サイズが1であれば、演算結果は不定となる。
上記以外のときには、このフィールドは使用されない。

NBCD

Negate Decimal with Extend・拡張付き10進ネゲート

操作：　0－（デスティネーション$_{10}$）－X→デスティネーション

アセンブラ・シンタックス：　NBCD <ea>

属性：　サイズ＝(バイト)

説明：　デスティネーション・オペランドと拡張ビットを0から減算します。操作は2進化10進算術演算を用いて行なわれます。デスティネーション・ロケーションにパックBCDの結果が入ります。この命令は拡張ビットがゼロの場合には、デスティネーションの10の補数を、そして拡張ビットがセットされていれば9の補数を生成します。これはバイト操作のみ可能です.

コンディション・コード：

X	N	Z	V	C
*	U	*	U	*

X－キャリ・ビットと同じ。

N－不定

Z－結果が0でなければセット、それ以外のときは変化しない。

V－不定

C－10進ボローが発生したらセット、それ以外のときはクリア。

注：通常コンディション・コードのZビットは、演算実行前にプログラムでセットされます。これにより、多倍精度演算を終了したとき、演算結果がゼロであるかどうかテストすることができます。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	0	実効アドレス					
										モード			レジスタ		

命令フィールド：

実効アドレス・フィールド――デスティネーション・オペランドを指定。次に示すとおり、データ可変アドレッシング・モードのみ可。

アドレッシング・モード	モード	レジスタ
Dn	000	reg. number : Dn
An	−	−
(An)	010	reg. number : An
(An)+	011	reg. number : An
−(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

アドレッシング・モード	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	−	−
(d_{16},PC)	−	−
(d_8,PC,Xn)	−	−
(bd,PC,Xn)	−	−
([bd,PC,Xn],od)	−	−
([bd,PC],Xn,od)	−	−

NEG

Negate・ネゲート

操作：　0－（デスティネーション）→デスティネーション

アセンブラ・シンタックス：　NEG < ea >

属性：　サイズ＝(バイト、ワード、ロング・ワード）

説明：　デスティネーション・オペランドを0から減算し、結果をデスティネーションに格納します。操作サイズは、バイト、ワード、ロング・ワードが指定できます。

コンディション・コード：

X	N	Z	V	C
*	*	*	*	*

X－キャリ・ビットと同じ。
N－結果が負であればセット、それ以外のときはクリア。
Z－結果が0であればセット、それ以外のときはクリア。
V－オーバフローが発生したときにセット、それ以外のときはクリア。
C－結果が0であればクリア、それ以外のときはセット。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
										実効アドレス					
0	1	0	0	0	1	0	0	サイズ		モード			レジスタ		

命令フィールド：

サイズ・フィールド──操作サイズを指定。
　00－バイト操作
　01－ワード操作
　10－ロング・ワード操作

実効アドレス・フィールド――デスティネーション・オペランドを指定。次に示すとおり、データ可変アドレッシング・モードのみ可。

ｱﾄﾞﾚｯｼﾝｸﾞ・ﾓｰﾄﾞ	モード	レジスタ
Dn	000	reg. number : Dn
An	–	–
(An)	010	reg. number : An
(An)＋	011	reg. number : An
−(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

ｱﾄﾞﾚｯｼﾝｸﾞ・ﾓｰﾄﾞ	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	–	–
(d_{16},PC)	–	–
(d_8,PC,Xn)	–	–
(bd,PC,Xn)	–	–
([bd,PC,Xn],od)	–	–
([bd,PC],Xn,od)	–	–

NEGX

Negate with Extend・拡張付きネゲート

操作：　0－（デスティネーション）－X→デスティネーション

アセンブラ・シンタックス：　NEGX <ea>

属性：　サイズ＝(バイト、ワード、ロング・ワード)

説明：　デスティネーション・オペランドと拡張ビットを0から減算し、結果をデスティネーション・ロケーションに格納します。操作サイズは、バイト、ワード、ロング・ワードが指定できます。

コンディション・コード：

X	N	Z	V	C
*	*	*	*	*

X－キャリ・ビットと同じ。

N－結果が負であればセット、それ以外のときはクリア。

Z－結果が0でなければクリア、それ以外のときは変化しない。

V－オーバフローが発生したときにセット、それ以外のときはクリア。

C－ボローが発生したときセット、それ以外のときはクリア。

注：通常コンディション・コードのZビットは、演算実行前にプログラムでセットされます。これにより、多倍精度演算を終了したとき、演算結果がゼロであるかどうかテストすることができます。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	サイズ		実効アドレス					
										モード			レジスタ		

命令フィールド：

サイズ・フィールド——操作サイズを指定。

00－バイト操作

01－ワード操作

10－ロング・ワード操作

実効アドレス・フィールド――デスティネーション・オペランドを指定。次に示すとおり、データ可変アドレッシング・モードのみ可。

アドレッシング・モード	モード	レジスタ
Dn	000	reg. number : Dn
An	–	–
(An)	010	reg. number : An
(An)+	011	reg. number : An
–(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

アドレッシング・モード	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	–	–
(d_{16},PC)	–	–
(d_8,PC,Xn)	–	–
(bd,PC,Xn)	–	–
([bd,PC,Xn],od)	–	–
([bd,PC],Xn,od)	–	–

NOP

None **・無操作**

操作：　なし

アセンブラ・シンタックス：　NOP

属性：　サイズなし

説明：　何も実行しません。プロセッサの状態は、プログラム・カウンタを除いて影響を受けません。プログラムの実行は、NOP 命令の次の命令から続行されます。NOP命令の実行は、保留されているバス・サイクルがすべて終了するまで完了しません。これによって、パイプラインの同期をとり、命令のオーバラップを防止します。詳細については、[3.7　NOP命令によるパイプラインの同期化」を参照してください。

コンディション・コード：　影響を受けない。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	0	1

NOT

Logical Complement・論理否定

操作：　～デスティネーション→デスティネーション

アセンブラ・シンタックス：　NOT <ea>

属性：　サイズ＝(バイト、ワード、ロング・ワード)

説明：　デスティネーション・オペランドの1の補数を計算し、結果をデスティネーション・ロケーションに格納します。操作サイズは、バイト、ワード、ロング・ワードを指定できます。

コンディション・コード：

X	N	Z	V	C
—	*	*	0	0

X－影響を受けない。
N－結果が負であればセット、それ以外のときはクリア。
Z－結果が0であればセット、それ以外のときはクリア。
V－常にクリア。
C－常にクリア。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	1	0	サイズ		実効アドレス					
										モード			レジスタ		

命令フィールド：

サイズ・フィールド――操作サイズを指定。
00－バイト操作
01－ワード操作
10－ロング・ワード操作

実効アドレス・フィールド――デスティネーション・オペランドを指定。次に示すとおり、データ可

ｱﾄﾞﾚｯｼﾝｸﾞ･ﾓｰﾄﾞ	モード	レジスタ
Dn	000	reg. number : Dn
An	–	–
(An)	010	reg. number : An
(An)+	011	reg. number : An
–(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

ｱﾄﾞﾚｯｼﾝｸﾞ･ﾓｰﾄﾞ	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	–	–
(d_{16},PC)	–	–
(d_8,PC,Xn)	–	–
(bd,PC,Xn)	–	–
([bd,PC,Xn],od)	–	–
([bd,PC],Xn,od)	–	–

OR

Inclusive OR Logical・論理和

操作：　ソースVデスティネーション→デスティネーション

アセンブラ・シンタックス：　OR < ea >, Dn
OR Dn, < ea >

属性：　サイズ＝(バイト、ワード、ロング・ワード)

説明：　ソース・オペランドとデスティネーション・オペランドの論理和をとり、結果をデスティネーション・ロケーションに格納します。操作サイズは、バイト、ワード、ロング・ワードが指定できます。アドレス・レジスタの内容をオペランドとして使用することはできません。

コンディション・コード：

X	N	Z	V	C
—	*	*	0	0

X－影響を受けない。
N－結果の最上ビットがセットされればセット、それ以外のときはクリア。
Z－結果が0であればセット、それ以外のときはクリア。
V－常にクリア。
C－常にクリア。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	レジスタ			Op モード			実効アドレス					
										モード			レジスタ		

命令フィールド：

レジスタ・フィールド――8つのデータ・レジスタのいずれかを指定。

Op モード・フィールド：

バイト	ワード	ロング・ワード	操　作
000	001	010	(< ea >) V (< Dn >) →< Dn >
100	101	110	(< Dn >) V (< ea >) →< ea >

実効アドレス・フィールド――指定されたロケーションがソース・オペランドの場合は、次に示すとおりデータ・アドレッシング・モードのみ可。

アドレッシング・モード	モード	レジスタ
Dn	000	reg. number : Dn
An	–	–
(An)	010	reg. number : An
(An)+	011	reg. number : An
–(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

アドレッシング・モード	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

指定されたロケーションがデスティネーション・オペランドの場合は、次に示すとおりメモリ可変アドレッシング・モードのみ可。

アドレッシング・モード	モード	レジスタ
Dn	–	–
An	–	–
(An)	010	reg. number : An
(An)+	011	reg. number : An
–(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

アドレッシング・モード	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	–	–
(d_{16},PC)	–	–
(d_8,PC,Xn)	–	–
(bd,PC,Xn)	–	–
([bd,PC,Xn],od)	–	–
([bd,PC],Xn,od)	–	–

注：1. デスティネーションがデータ・レジスタの場合、デスティネーション<ea>モードではなく、デスティネーションDnモードを使用して指定しなければなりません。

2. ほとんどのアセンブラは、ソースがイミディエイト・データのときにはORIを使用します。

ORI

Inclusive OR Immediate・イミディエイト論理和

操作：　イミディエイト・データVデスティネーション→デスティネーション

アセンブラ・シンタックス：　ORI #<data>,<ea>

属性：　サイズ＝(バイト、ワード、ロング・ワード)

説明：　イミディエイト・データとデスティネーション・オペランドとの論理和をとり、結果をデスティネーション・ロケーションに格納します。操作サイズは、バイト、ワード、ロング・ワードが指定できます。イミディエイト・データのサイズは操作サイズと同じです。

コンディション・コード：

X	N	Z	V	C
—	*	*	0	0

X－影響を受けない。

N－結果の最上位ビットがセットされればセット、それ以外のときはクリア。

Z－結果が0であればセット、それ以外のときはクリア。

V－常にクリア。

C－常にクリア。

命令フォーマット：

<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">0</td><td colspan="2" rowspan="2">サイズ</td><td colspan="6">実効アドレス</td></tr>
<tr><td colspan="3">モード</td><td colspan="3">レジスタ</td></tr>
<tr><td colspan="8">ワード・データ（16ビット）</td><td colspan="8">バイト・データ（8ビット）</td></tr>
<tr><td colspan="16">ロング・データ（32ビット）</td></tr>
</table>

命令フィールド：

サイズ・フィールド――操作サイズを指定。

00－バイト操作

01－ワード操作

10－ロング・ワード操作

実効アドレス・フィールド――デスティネーション・オペランドを指定。次に示すとおり、データ可変アドレッシング・モードのみ可。

アドレッシング・モード	モード	レジスタ
Dn	000	reg. number : Dn
An	–	–
(An)	010	reg. number : An
(An)+	011	reg. number : An
–(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

アドレッシング・モード	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	–	–
(d_{16},PC)	–	–
(d_8,PC,Xn)	–	–
(bd,PC,Xn)	–	–
([bd,PC,Xn],od)	–	–
([bd,PC],Xn,od)	–	–

イミディエイト・フィールド――（命令直後のデータ）

サイズ＝00　データはイミディエイト・ワードの下位バイト。

サイズ＝01　データはイミディエイト・ワード全体。

サイズ＝10　データは次の2つのイミディエイト・ワード。

ORI to CCR

Inclusive OR Immediate to Condition Codes・コンディション・コードとのイミディエイト論理和

操作：　ソースＶ CCR→CCR

アセンブラ・シンタックス：　ORI #<data>, CCR

属性：　サイズ＝(バイト)

説明：　イミディエイト・オペランドとコンディション・コードの論理和をとり、結果をコンディション・コード・レジスタ(ステータス・レジスタの下位バイト)に格納します。

コンディション・コード：

X	N	Z	V	C
*	*	*	*	*

X－イミディエイト・オペランドのビット4が1であればセット、それ以外のときは変化しない。

N－イミディエイト・オペランドのビット3が1であればセット、それ以外のときは変化しない。

Z－イミディエイト・オペランドのビット2が1であればセット、それ以外のときは変化しない。

V－イミディエイト・オペランドのビット1が1であればセット、それ以外のときは変化しない。

C－イミディエイト・オペランドのビット0が1であればセット、それ以外のときは変化しない。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0
0	0	0	0	0	0	0	0	バイト・データ（8ビット）							

ORI to SR

Inclusive OR Immediate to the Status Register(Privileged Instruction)・
ステータス・レジスタとのイミディエイト論理和(特権命令)

操作：　スーパバイザ状態ではソースV SR→SR
　　　　ユーザ状態ではTRAP

アセンブラ・シンタックス　ORI #<data>, SR

属性：　サイズ=(ワード)

説明：　イミディエイト・オペランドとステータス・レジスタの内容の論理和をとり、結果をステータス・レジスタに格納します。ステータス・レジスタに実装されている全ビットが影響を受けます。

コンディション・コード：

X	N	Z	V	C
*	*	*	*	*

X－イミディエイト・オペランドのビット4が1であればセット、それ以外のときは変化しない。
N－イミディエイト・オペランドのビット3が1であればセット、それ以外のときは変化しない。
Z－イミディエイト・オペランドのビット2が1であればセット、それ以外のときは変化しない。
V－イミディエイト・オペランドのビット1が1であればセット、それ以外のときは変化しない。
C－イミディエイト・オペランドのビット0が1であればセット、それ以外のときは変化しない。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0
ワード・データ（16ビット）															

PACK

Pack・パック

操作：　ソース(アンパック BCD)＋調整値→デスティネーション(パック BCD)

アセンブラ・シンタックス：　PACK －(Ax), －(Ay), #<調整値>
PACK Dx, Dy, #<調整値>

属性：　サイズなし

説明：　各2バイトの下位4ビットを調整し、1バイトにパックします。

オペランドが両方ともデータ・レジスタの場合、ソース・レジスタにある値に調整値が加算されます。中間結果のビット [11：8] と [3：0] が連結されて、デスティネーション・レジスタのビット [7：0] に置かれます。デスティネーション・レジスタの残りの部分は影響を受けません。

ソース（Dx）：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
x	x	x	x	a	b	c	d	x	x	x	x	e	f	g	h

調整ワードの加算：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
16ビット拡張															

結果：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
x'	x'	x'	x'	a'	b'	c'	d'	x'	x'	x'	x'	e'	f'	g'	h'

デスティネーション（Dy）：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u	u	u	u	u	u	u	u	a'	b'	c'	d'	e'	f'	g'	h'

プリデクリメントのアドレッシング・モードが指定されているときには、ソースから2バイトがフェッチされて、連結されます。連結されたバイトに調整ワードが加算されます。各バイトのビット [3：0] が抽出されます。これらの8ビットが連結され新しいバイトが形成されて、デスティネーションに書き込まれます。

ソース（Ax）：

7	6	5	4	3	2	1	0
x	x	x	x	a	b	c	d
x	x	x	x	e	f	g	h

連結されたワード：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
x	x	x	x	a	b	c	d	x	x	x	x	e	f	g	h

調整ワードの加算：

デスティネーション（Ay）：

コンディション・コード：　影響を受けない。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	レジスタDy／Ay			1	0	1	0	0	R/M	レジスタDx／Ax		
16ビット拡張：調整															

命令フィールド：

レジスタDy/Ayフィールド――デスティネーション・レジスタを指定。

R/M＝0　データ・レジスタを指定。

R/M＝1　プリデクリメント・アドレッシング・モードのためのアドレス・レジスタを指定。

R/Mフィールド――オペランドのアドレッシング・モードを指定。

0－データ・レジスタとデータ・レジスタの操作。

1－メモリとメモリの操作。

レジスタDx/Axフィールド――ソース・レジスタを指定。

R/M＝0　データ・レジスタを指定。

R/M＝1　プリデクリメント・アドレッシング・モードのアドレス・レジスタを指定。

調整フィールド――ソース・オペランドに加算されるイミディエイト・データ・ワード。このワードがゼロでASCIIまたはEBCDICコードのパックを行なう。他のコードに対しては、別のコードを使用することができる。

PEA

Push Effective Address・実効アドレスのプッシュ

操作：　SP－4→SP；〈ea〉→（SP）

アセンブラ・シンタックス：　PEA　＜ea＞

属性：　サイズ＝（ロング・ワード）

説明：　実効アドレスを計算し、スタックにプッシュします。実効アドレスはロング・ワード・アドレスです。

コンディション・コード：　影響を受けない。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	1	実効アドレス モード			実効アドレス レジスタ		

命令フィールド：

実効アドレス・フィールド――スタックにプッシュするアドレスを指定。次に示すとおり、制御アドレッシング・モードのみ可。

アドレッシング・モード	モード	レジスタ
Dn	－	－
An	－	－
(An)	010	reg. number : An
(An)＋	－	－
－(An)	－	－
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

アドレッシング・モード	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	－	－
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

PFLUSH

Flush Entry in the ATC(Privileged Instruction)・
ATC内のエントリのフラッシュ(特権命令)

操作：　スーパバイザ状態ではデスティネーション・アドレスに対するATCエントリを無効にする。
ユーザ状態ではTRAP

アセンブラ・シンタックス：　PFLUSHA
PFLUSH < fc >, # < mask >
PFLUSH < fc >, # < mask >, < ea >

属性：　サイズなし

説明：　ATCエントリを無効にします。この命令には3つのフォームがあります。PFLUSHA命令はすべてのエントリを無効にします。命令がファンクション・コード< fc >とマスク< mask >を指定すると、その命令は選択された1つまたは複数のファンクション・コードに対応するすべてのエントリを無効にします。命令が実効アドレス< ea >も指定するときには、命令は選択された各ファンクション・コードにある実効アドレス・エントリに対応するページ・ディスクリプタも無効にします。
< mask >オペランドには3ビットのファンクション・コードに対応する3ビットがあります。マスクの各ビットが1にセットされているときは、< fc >オペランドの対応するビットが操作に適用されることを示します。マスクで0になっている各ビットは、< fc >およびファンクション・コードのビットを無視することを示します。たとえば、マスク・オペランド100では、命令を< fc >オペランドの最上位ビットだけと考えることができます。< fc >オペランドが001の場合は、ファンクション・コード000、001、010および011が選択されます。
< fc >オペランドは次のいずれかの方法で指定されます。

1. イミディエイト――コマンド・ワードの3ビット
2. データ・レジスタ――命令で指定されるデータ・レジスタの最下位3ビット
3. ソース・ファンクション・コード・レジスタ
4. デスティネーション・ファンクション・コード・レジスタ

MMUの情報については、「第9章　メモリ管理ユニット」を参照してください。

コンディション・コード：　影響を受けない。

MMUSR：　影響を受けない。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	0	0	実効アドレス モード			実効アドレス レジスタ		
0	0	1	モード			0	0	マスク			FC				

第3章 命令セット

命令フィールド：

実効アドレス・フィールド――制御可変アドレスを指定。このアドレスのATCエントリは無効になる。有効なアドレッシング・モードは次のとおり。

ｱﾄﾞﾚｯｼﾝｸﾞ･ﾓｰﾄﾞ	モード	レジスタ
Dn	−	−
An	−	−
(An)	010	reg. number : An
(An)+	−	−
−(An)	−	−
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

ｱﾄﾞﾚｯｼﾝｸﾞ･ﾓｰﾄﾞ	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	−	−
(d_{16},PC)	−	−
(d_8,PC,Xn)	−	−
(bd,PC,Xn)	−	−
([bd,PC,Xn],od)	−	−
([bd,PC],Xn,od)	−	−

注：アドレス・フィールドはメモリ管理ユニット(MMU)に、PFLUSHオペランドが存在する位置を記述する実効アドレスではなく、ATCからフラッシュする実効アドレスを与えなければなりません。たとえば、システム・スタックの先頭に一時的に格納される論理アドレスに対応するATCエントリをフラッシュするには、'PFLUSH [(SP)]' を使用します。これは、'PFLUSH (SP)' がシステム・スタックにマッピングされるATCエントリを無効にするためです(すなわち、MMUに渡される実効アドレスはシステム・スタックの実効アドレスであって、スタックの先頭に位置するオペランドで形成される実効アドレスではありません)。

モード・フィールド――フラッシュ操作のタイプを示す。

001－すべてのエントリをフラッシュする。

100－ファンクション・コードによってのみフラッシュする。

110－ファンクション・コードと実効アドレスによってフラッシュする。

マスク・フィールド――ファンクション・コードを選択するマスク。マスクの1は適用されるビットに対応する。0は無視するビットである。モードが001のときには、マスクは000でなければならない。

FCフィールド――フラッシュするエントリのファンクション・コード。モードが001のときには、FCは00000でなければならない。

10XXX－ファンクション・コードはビットXXXとして指定される。

01DDD－ファンクション・コードはデータ・レジスタDDDのビット2：0として指定される。

00000 －ファンクション・コードはSFCレジスタとして指定される。

00001 －ファンクション・コードはDFCレジスタとして指定される。

PLOAD

Load an Entry into the ATC(Privileged Instruction)・

ATCへのエントリのロード(特権命令)

操作：　スーパバイザ状態ではエントリ→ATC
　　　　ユーザ状態ではTRAP

アセンブラ・シンタックス：　PLOADR < function　code >, < ea >
　　　　PLOADDW < function　code >, < ea >

属性：　サイズなし

説明：　ATCで指定された実効アドレスをサーチします。また、指定された実効アドレスに対応するディスクリプタの変換テーブルもサーチします。新しいエントリは、MC68030がそのアドレスへのアクセスを試みたかのように生成されます。サーチの一部として、使用されているビットおよび変更されたビットを適宜セットします。この命令は、変換制御(TC)レジスタのEビットの値または$\overline{\text{MMUDIS}}$信号の状態に関係なく実行されます。詳細については、「9. 5. 2　一般的なテーブル・サーチ」を参照してください。

< function　code >オペランドは次のいずれかの方法で指定されます。

1. イミディエイト――コマンド・ワードの3ビット
2. データ・レジスタ――命令で指定されるデータ・レジスタの最下位3ビット
3. ソース・ファンクション・コード・レジスタ
4. デスティネーション・ファンクション・コード・レジスタ

PLOADRにより変換テーブルのUビットは、リード・アクセスの場合と同様にセットされます。PLOADWを実行すると、ライト・アクセスの場合と同様にセットされます。

MMUの情報については、「第9章　メモリ管理ユニット」を参照してください。

コンディション・コード：　影響を受けない。

MMUSR：　影響を受けない。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
										実効アドレス					
1	1	1	1	0	0	0	0	0	0	モード			レジスタ		
0	0	1	0	0	0	R/W	0	0	0	0	FC				

命令フィールド：

実効アドレス・フィールド――変換する論理アドレスを指定する。次に示すとおり、制御可変アドレッシング・モードのみ可。

アドレッシング・モード	モード	レジスタ
Dn	−	−
An	−	−
(An)	010	reg. number : An
(An)＋	−	−
−(An)	−	−
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

アドレッシング・モード	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	−	−
(d_{16},PC)	−	−
(d_8,PC,Xn)	−	−
(bd,PC,Xn)	−	−
([bd,PC,Xn],od)	−	−
([bd,PC],Xn,od)	−	−

＜ea＞フィールドは、変換するアドレス・ロケーションの実効アドレスではなく、変換する実効アドレスを指定する。たとえば、システム・スタックに格納されている論理アドレスに対応するATCエントリをロードするための正しい命令はPLOAD([SP])である。PLOAD(SP)はシステム・スタック・ロケーションの内容に対するATCエントリではなく、システム・スタック・ロケーションに対するATCエントリをロードする。

R/Wフィールド――テーブル・サーチに使用するアクセスのタイプを指定する。

0－ライト

1－リード

FCフィールド――ロードするエントリに対応するファンクション・コード。

10XXX－ファンクション・コードはビットXXXとして指定される。

01DDD－ファンクション・コードはデータ・レジスタDDDのビット2：0として指定される。

00000 －ファンクション・コードはSFCレジスタとして指定される。

00001 －ファンクション・コードはDFCレジスタとして指定される。

PMOVE

Move to/from MMU Registers(Privileged Instruction) •
MMUレジスタ転送（特権命令）

操作：　スーパバイザ状態では（ソース）→MRnまたはMRn→（デスティネーション）

アセンブラ・シンタックス：　PMOVE MRn, < ea >
PMOVE < ea >, MRn
PMOVEFD < ea >, MRn

属性：　サイズ=(ワード、ロング・ワード、クワッド・ワード)

説明：　ソース実効アドレスの内容を指定されたMMUレジスタに、またはMMUレジスタの内容をデスティネーション実効アドレスに転送します。

この命令はCPUルート・ポインタ(CRP)およびスーパバイザ・ルート・ポインタ(SRP)に対してはクワッド・ワード(8バイト)操作を行ないます。変換制御レジスタ(TC)および透過変換レジスタ(TT0およびTT1)に対してはロング・ワード操作です。また、MMUステータス・レジスタ(MMUSR)に対してはワード操作になります。

この命令のPMOVEFD(フラッシュがディセーブルになったPMOVE)フォームは、FDビットをセットして、SRP、CRP、TT0、TT1またはTCレジスタ(ただし、MMUSRを除く)に新しい値がロードされたときに、ATCのフラッシングをディセーブルにします。

次のレジスタに書込みを行なっても同様な二次的効果があります。

CPUルート・ポインタ

FDビットがゼロのときに、ATCをフラッシュします。ルート・ポインタ・ディスクリプタに対してオペランド値が無効の場合、命令はオペランドをCRPに転送した後、MMUコンフィギュレーション・エラー例外を発生します。

スーパバイザ・ルート・ポインタ

FDビットがゼロのときに、ATCをフラッシュします。オペランドの値がルート・ポインタ・ディスクリプタとして無効の場合、命令はオペランドをSRPに転送した後、MMUコンフィギュレーション・エラー例外を発生します。

変換制御

FDビットがゼロのときに、ATCをフラッシュします。Eビットの値が1の場合には、PSとTIxフィールドについて一貫性チェックが実行されます。チェックが不合格の場合、命令はオペランドをTCに転送したあと、MMUコンフィギュレーション例外を発生します。チェックに合格したときは、TCレジスタにはEをセットしたオペランドがロードされます。

透過変換

FDビットがゼロのときに、ATCをフラッシュします。書き込まれたEビットに従って、透過変換レジスタをイネーブルまたはディセーブルします。Eビットが1にセットされている場合は、透過変換レジスタがイネーブルされます。Eビットが0の場合は、レジスタがディセーブルされます。

MMUの情報については、「第9章　メモリ管理ユニット」を参照してください。

コンディション・コード：　影響を受けない。

MMUSR：　影響を受けない(MMUSRがデスティネーション・オペランドとして指定されていない場合)。

命令フォーマット(SRP、CRP、およびTCレジスタ):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	0	0	実効アドレス モード			実効アドレス レジスタ		
0	1	0	P REG			R/W	FD	0	0	0	0	0	0	0	0

命令フィールド(SRP、CRP、およびTCレジスタ):

実効アドレス・フィールド――転送のメモリ・ロケーションを指定する。制御可変アドレッシング・モードのみ可。

アドレッシング・モード	モード	レジスタ
Dn	–	–
An	–	–
(An)	010	reg. number : An
(An)+	–	–
–(An)	–	–
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

アドレッシング・モード	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#<data>	–	–
(d_{16},PC)	–	–
(d_8,PC,Xn)	–	–
(bd,PC,Xn)	–	–
([bd,PC,Xn],od)	–	–
([bd,PC],Xn,od)	–	–

P Regフィールド――MMUレジスタを指定。

000 – TC

010 – SRP

011 – CRP

R/Wフィールド――転送の方向を指定。

0 – メモリからMMUレジスタ

1 – MMUレジスタからメモリ

FDフィールド――MMUレジスタへの書込み時におけるATCのフラッシングをディセーブル。

0 – ATCがフラッシュされる。

1 – ATCがフラッシュされない。

命令フォーマット(MMUSR):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	0	0	実効アドレス モード			実効アドレス レジスタ		
0	1	1	0	0	0	R/W	0	0	0	0	0	0	0	0	0

命令フィールド(MMUSR)：

実効アドレス・フィールド――転送のメモリ・ロケーションを指定。SRPレジスタで示す制御可変アドレッシング・モードを適用。

R/Wフィールド――転送の方向を指定。

0－メモリからMMUステータス・レジスタ

1－MMUステータス・レジスタからメモリ

注：MC68851のアセンブラのシンタックスはMMUのステータス・レジスタに対し、記号PSRを使用しています。

命令フォーマット(TTレジスタに対応)：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	0	0	実効アドレス モード			レジスタ		
0	0	0	P REG			R/W	FD	0	0	0	0	0	0	0	0

命令フィールド(TTレジスタに対応)：

実効アドレス・フィールド――転送のメモリ・ロケーションを指定。SRPレジスタで示す制御可変アドレッシング・モードを適用。

P Regフィールド――TTレジスタを指定。

010－透過変換レジスタ0

011－透過変換レジスタ1

R/Wフィールド――転送の方向を指定。

0－メモリからMMUステータス・レジスタ

1－MMUステータス・レジスタからメモリ

FDフィールド――ATCのフラッシングをディセーブル。

0－ATCがフラッシュされる。

1－ATCがフラッシュされない。

PTEST

Test a Logical Address(Privileged Instruction)・

論理アドレスのテスト(特権命令)

操作：　スーパバイザ状態では論理アドレス・ステータス→MMUSR
ユーザ状態ではTRAP

アセンブラ・シンタックス：　PTESTR <function code>, <ea>, #<level>
PTESTR <function code>, <ea>, #<level>, An
PTESTW <function code>, <ea>, #<level>
PTESTW <function code>, <ea>, #<level>, An

属性：　サイズなし

説明：　この命令は指定されたレベルのATCまたは変換テーブルで、<ea>フィールドに対応する変換ディスクリプタをサーチし、ディスクリプタのステータスに従ってMMUステータス・レジスタ(MMUSR)のビットをセットします。オプションとして、PTESTはサーチ中に最後にアクセスされたテーブル・エントリの物理アドレスを格納します。PTEST命令は、ATCまたは変換テーブルをサーチしてステータス情報を取得しますが、変換テーブルおよびATCの“使用”、“変更”ビットは変更しません。レベル・オペランドがゼロのときは、リードまたはライト・アクセスのいずれかの透過変換によって、PTESTRおよびPTESTWの操作が異なる結果を返すことになります。

<function code>オペランドは次のいずれかの方法で指定します。

1. イミディエイト――コマンド・ワードの3ビット
2. データ・レジスタ――命令で指定されるデータ・レジスタの最下位3ビット
3. ソース・ファンクション・コード・レジスタ
4. デスティネーション・ファンクション・コード・レジスタ

実効アドレスはテストを行なうアドレスです。<level>オペランドはサーチのレベルを指定します。レベル0はATCだけをサーチすることを指定します。値1～7は変換テーブルだけをサーチすることを指定します。サーチは指定されたレベルで終了します。レベル0テストはゼロ以外のレベル番号でのテストと同じMMUSR値を返しません。

命令の実行は要求されたレベルまで、あるいは次の条件の1つが検出されるまで続きます。

- 無効ディスクリプタ
- 限界違反
- バス・エラーのアサート(物理バス・エラー)

命令は連続したテーブル・エントリをアクセスするときはステータスを蓄積します。

命令がアドレス・レジスタ・オペランドでATCサーチを指定すると、MC68030はFライン未実装命令例外の処理を行ないます。

アドレス・レジスタ・パラメータが変換テーブル・サーチに指定されている場合は、正常にフェッチを行なった最後のディスクリプタの物理アドレスがアドレス・レジスタにロードされます。ディスクリプタは、ディスクリプタのすべてのポインタをMC68030が、バス・サイクルの異常バス・ターミネーションを起こさないで読み出すことができる場合にかぎり、“正常に”フェッチされます。使用されたルート・ポインタのDTフィールドが“ページ・ディスクリプタ”を示す場合は、アドレス$0が返されます。ロング・ディスクリプタに対しては、最初のロング・ワードのアドレスが返されます。ディスクリプタのサイズ(ショートまたはロング)は返されず、変換テーブルの情報から確定しなければなりません。

MMUの情報については、「第9章　メモリ管理ユニット」を参照してください。

コンディション・コード：　影響を受けない。

MMUSR：

B	L	S		W	I	M			T				N
*	*	*	0	*	*	*	0	0	0	0	0	0	*

MMUステータス・レジスタはサーチの結果を保持している。ATCサーチに対するMMUSRの各フィールドの値は次のとおり。

MMUSRビット	PTEST, レベル0	PTEST, レベル1～7
バス・エラー（B）	このビットは指定された論理アドレスに対応するATCエントリで、バス・エラー・ビットがセットされている場合にセットされる。	このビットはPTEST命令のテーブル・サーチ中にバス・エラーが発生した場合にセットされる。
リミット（L）	このビットはクリアされる。	このビットはテーブル・サーチ中にインデックスがリミットを超えた場合にセットされる。
スーパバイザ違反（S）	このビットはクリアされる。	このビットは、サーチ中に出会ったロング（S）フォーマット・テーブル・ディスクリプタまたはロング・フォーマット・ページ・ディスクリプタのSビットがセットされており、かつPTEST命令で指定されたファンクション・コードのFC2ビットが1と等しくない場合にセットされる。Iビットがセットされている場合、Sビットは未定義。
ライト・プロテクト（W）	このビットはATCエントリのWPビットがセットされている場合にセットされる。Iビットがセットされている場合は未定義。	このビットはテーブル・サーチ中に出会ったディスクリプタまたはページ・ディスクリプタのWPビットがセットされている場合にセットされる。Iビットがセットされている場合、Wビットは未定義。
無効（I）	このビットは無効な変換を示す。Iビットは、指定された論理アドレスに対する変換がATCにない場合、または対応するATCエントリのBビットがセットされている場合にセットされる。	このビットは無効な変換を示す。Iビットはサーチ中に出会ったテーブルまたはページ・ディスクリプタのDTフィールドが無効になっている場合、またはテーブル・サーチ中にMMUSRのBまたはLビットがセットされている場合にセットされる。
修正（M）	このビットは指定されたアドレスに対応するATCエントリの修正ビットがセットされている場合にセットされる。Iビットがセットされている場合は未定義。	このビットは指定されたアドレスのページ・ディスクリプタの修正ビットがセットされている場合にセットされる。Iビットがセットされている場合は未定義。
透過（T）	このビットは透過変換レジスタ（TT0またはTT1）のいずれか（または両方）で一致があった場合にセットされる。Iビットがセットされている場合は未定義。	このビットはゼロにセットされる。
レベル数（N）	この3ビット・フィールドはゼロにクリアされる。	この3ビット・フィールドにはサーチ中にアクセスされる実際のテーブル番号が入っている。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	0	0	実効アドレス モード			実効アドレス レジスタ		
1	0	0	レベル			R/W	A	REG			FC				

命令フィールド：

実効アドレス・フィールド――テストする論理アドレスを指定。次に示すように、可変制御アドレッシング・モードのみ可。

ｱﾄﾞﾚｯｼﾝｸﾞ・ﾓｰﾄﾞ	モード	レジスタ
Dn	–	–
An	–	–
(An)	010	reg. number : An
(An)+	–	–
–(An)	–	–
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

ｱﾄﾞﾚｯｼﾝｸﾞ・ﾓｰﾄﾞ	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	–	–
(d_{16},PC)	–	–
(d_8,PC,Xn)	–	–
(bd,PC,Xn)	–	–
([bd,PC,Xn],od)	–	–
([bd,PC],Xn,od)	–	–

レベル・フィールド――テーブルでサーチする最も大きい番号を指定。このフィールドが0のときは、Aフィールドおよびフィールドも0でなければならない。レベル・フィールドが0でAフィールドが0でないとき、この命令はFライン例外を発生する。

R/Wフィールド――リードまたはライト・バス・サイクル(MC68030 MMUでは相違はない)のシミュレーションを指定。

0－ライト

1－リード

Aフィールド――アドレッシング・レジスタ・オプションを指定する。

0－アドレス・レジスタなし

1－Regフィールドで指定されたアドレス・レジスタで最後にサーチされたディスクリプタのアドレスを返す。

Regフィールド――命令のアドレス・レジスタを指定する。Aフィールドが0のとき、このフィールドも0でなければならない。

FCフィールド――テストするアドレスのファンクション・コード。

10XXX－ファンクション・コードはビットXXXで指定。

01DDD－ファンクション・コードはデータ・レジスタDDDのビット2：0で指定。

00000　－ファンクション・コードはSFCレジスタで指定。

00001　－ファンクション・コードはDFCレジスタで指定。

RESET

Reset External Devices(Privileged Instruction)・外部デバイスのリセット(特権命令)

操作：　スーパバイザ状態では$\overline{\text{RESET}}$ラインをアサート
ユーザ状態ではTRAP

アセンブラ・シンタックス：　RESET

属性：　サイズなし

説明：　512クロックの間、リセット・ラインをアサートし、すべての外部デバイスをリセットします。プロセッサ状態はプログラム・カウンタを除いて影響を受けることはなく、プログラムの実行は次の命令から続行されます。

コンディション・コード：　影響を受けない。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	0	0

ROL / ROR

Rotate (Without Extend) • 拡張なしローテイト

操作：　デスティネーション Rotated by <カウント>→デスティネーション

アセンブラ・シンタックス：　ROd Dx, Dy
ROd #<data>, Dy
ROd <ea>
ここでdは方向で、L（左）またはR（右）

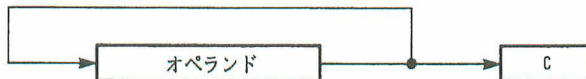
属性：　サイズ＝(バイト、ワード、ロング・ワード)

説明：　デスティネーション・オペランドのビットを指定方向(右または左)にローテイトします。ローテイトに拡張ビットは含まれません。レジスタのローテイト・カウントは次の2とおりの方法で指定されます。

1. イミディエイト——ローテイト・カウント(1-8)を命令の中で指定します。
2. レジスタ——ローテイト・カウント(モジュロ64)は命令で指定するデータ・レジスタにあります。

レジスタ・デスティネーションの操作サイズは、バイト、ワード、ロング・ワードが指定できます。ただし、メモリ内容<ea>は、1ビットしかローテイトできず、オペランド・サイズもワードに限定されます。

ROL命令は、オペランドのビットを左にローテイトします。ローテイト・カウントでローテイトする位置数が決まります。最上位から送り出されたビットはキャリ・ビットと最下位ビットの両方に入ります。

ROR命令は、オペランドのビットを右にローテイトします。ローテイト・カウントでローテイトする位置数が決まります。最上位から送り出されたビットはキャリ・ビットと最下位ビットの両方に入ります。

コンディション・コード：

X	N	Z	V	C
—	*	*	0	*

X－影響を受けない。
N－結果の最上位ビットがセットされればセット、それ以外のときはクリア。
Z－結果が0であればセット、それ以外のときはクリア。
V－常にクリア。
C－オペランドから最後に送り出されたビットに従ってセット、ローテイト・カウントが0のときはクリア。

命令フォーマット(レジスタ・ローテイト)：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	カウント/レジスタ			dr	サイズ		i/r	1	1	レジスタ		

命令フィールド(レジスタ・ローテイト)：

カウント/レジスタ・フィールド
　i/r＝0　ローテイト・カウントはこのフィールドにある。1-7の値は1-7、0は8を表わす。
　i/r＝1　このフィールドはローテイト・カウント(モジェロ64)をもつデータ・レジスタを指定する。
dr フィールド——ローテイトの方向を指定。
　0－右ローテイト
　1－左ローテイト
サイズ・フィールド——操作サイズを指定。
　00－バイト操作
　01－ワード操作
　10－ロング・ワード操作
i/r フィールド——ローテイト・カウントのロケーションを指定。
　i/r＝0　ローテイト・カウントをイミディエイト・データで指定。
　i/r＝1　ローテイト・カウントをレジスタで指定。
レジスタ・フィールド——ローテイトするデータ・レジスタを指定。

命令フォーマット(メモリ・ローテイト)：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	1	1	dr	1	1	実効アドレス モード			レジスタ		

命令フィールド(メモリ・ローテイト)：

dr フィールド——ローテイト方向を指定。

0－右ローテイト

1－左ローテイト

実効アドレス・フィールド——ローテイトするオペランドを指定。次に示すとおり、メモリ可変アドレッシング・モードのみ可。

ｱﾄﾞﾚｯｼﾝｸﾞ･ﾓｰﾄﾞ	モード	レジスタ
Dn	－	－
An	－	－
(An)	010	reg. number : An
(An)＋	011	reg. number : An
－(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

ｱﾄﾞﾚｯｼﾝｸﾞ･ﾓｰﾄﾞ	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	－	－
(d_{16},PC)	－	－
(d_8,PC,Xn)	－	－
(bd,PC,Xn)	－	－
([bd,PC,Xn],od)	－	－
([bd,PC],Xn,od)	－	－

ROXL / ROXR

Rotate (with Extend) • 拡張付きローテイト

操作：　デスティネーション Rotated with X by <カウント>→デスティネーション

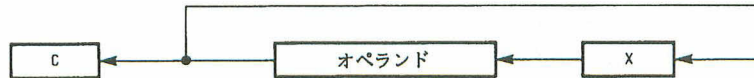
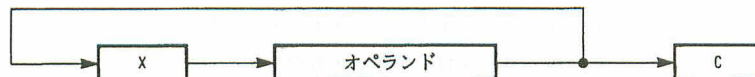
アセンブラ・シンタックス：　ROXd Dx, Dy
ROXd #<data>, Dy
ROXd <ea>
ここでdは方向で、L（左）またはR（右）

属性：　サイズ＝(バイト、ワード、ロング・ワード)

説明：　オペランドのビットを指定方向(右または左)にローテイトします。ローテイトに拡張ビットが含まれます。レジスタのローテイト・カウントは次の2とおりの方法で指定されます。

1. イミディエイト――ローテイト・カウント(1～8)は命令の中で指定します。
2. レジスタ――ローテイト・カウント(モジュロ64)は命令で指定するデータ・レジスタにあります。

レジスタ・デスティネーションに対する操作サイズは、バイト、ワード、ロング・ワードが指定できます。ただし、メモリ内容<ea>の場合は1ビットしかローテイトできず、オペランド・サイズもワードに限定されます。

ROXL命令は、オペランドのビットを左にローテイトします。ローテイト・カウントでローテイトする位置数が決まります。最上位ビットから送り出されたビットはキャリ・ビットと拡張ビットの両方に入り、拡張ビットの以前の値が最下位ビットに送り出されます。

ROXR命令は、オペランドのビットを右にローテイトします。ローテイト・カウントでローテイトする位置数が決まります。最下位ビットから送り出されたビットはキャリ・ビットと拡張ビットの両方に入り、拡張ビットの以前の値が最上位ビットに送り出されます。

コンディション・コード：

X	N	Z	V	C
*	*	*	0	*

X－オペランドから最後に送り出されたビットに従ってセット、ローテイト・カウントが0のときは影響を受けない。

N－結果の最上位ビットがセットされればセット、それ以外のときはクリア。

Z－結果が0であればセット、それ以外のときはクリア。

V－常にクリア。

C－オペランドから最後に送り出されたビットに従ってセット、ローテイト・カウントが0のときは拡張ビットの値をセット。

命令フォーマット(レジスタ・ローテイト)：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	カウント/レジスタ			dr	サイズ		i/r	1	0	レジスタ		

命令フィールド(レジスタ・ローテイト)：

カウント/レジスタ・フィールド

i/r＝0　ローテイト・カウントはこのフィールドに含まれる。1-7の値は1-7を、0は8を表わす。

i/r＝1　ローテイト・カウント(モジェロ64)は、このフィールドで指定するデータ・レジスタにある。

drフィールド──ローテイトの方向を指定。

0－右ローテイト

1－左ローテイト

サイズ・フィールド──操作サイズを指定。

00－バイト操作

01－ワード操作

10－ロング・ワード操作

i/rフィールド──ローテイト・カウントのロケーションを指定。

i/r＝0　イミディエイト・ローテイト・カウント

i/r＝1　レジスタ・ローテイト・カウント

レジスタ・フィールド──ローテイトするデータ・レジスタを指定。

命令フォーマット(メモリ・ローテイト)：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
										実効アドレス					
										モード			レジスタ		
1	1	1	0	0	1	0	dr	1	1						

命令フィールド(メモリ・ローテイト)：

drフィールド——ローテイト方向を指定。

0－右ローテイト

1－左ローテイト

実効アドレス・フィールド——ローテイトするオペランドを指定。次に示すとおり、メモリ可変アドレッシング・モードのみ可。

ｱﾄﾞﾚｯｼﾝｸﾞ・ﾓｰﾄﾞ	モード	レジスタ
Dn	－	－
An	－	－
(An)	010	reg. number : An
(An)＋	011	reg. number : An
－(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

ｱﾄﾞﾚｯｼﾝｸﾞ・ﾓｰﾄﾞ	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#<data>	－	－
(d_{16},PC)	－	－
(d_8,PC,Xn)	－	－
(bd,PC,Xn)	－	－
([bd,PC,Xn],od)	－	－
([bd,PC],Xn,od)	－	－

RTD

Return and Deallocate ・リターンおよび、パラメータの割当て解除

操作：　(SP)→PC；SP＋4＋d→SP

アセンブラ・シンタックス：　RTD　#<displacement>

属性：　サイズなし

説明：　プログラム・カウンタをスタックからプルし、それに符号拡張された16ビット・ディスプレースメント値を付加します。以前のプログラム・カウンタの値は失われます。

コンディション・コード：　影響を受けない。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	0	0
ディスプレースメント（16ビット）															

命令フィールド：

ディスプレースメント・フィールド――符号拡張してスタック・ポインタに加えられる2の補数の整数を指定。

RTE

Return from Exception (Privileged Instruction)・例外処理からのリターン(特権命令)

操作：　スーパバイザ状態では(SP)→SR；SP＋2→SP；(SP)→PC；SP＋4→SP；
状態をリストアし、(SP)に従ってスタックの割当てを解除する。
ユーザ状態ではTRAP

アセンブラ・シンタックス：　RTE

属性：　サイズなし

説明：　スタックの先頭にある例外スタック・フレームにあるプロセッサ状態情報をロードします。この命令はフォーマット/オフセット・ワードの中のスタック・フォーマット・フィールドを調べ、どれだけの情報をリストアしなければならないかを決定する。

コンディション・コード：　スタックからのステータス・レジスタ中のコンディション・コードの内容に従ってセットされる。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	1	1

フォーマット/オフセット・ワード(スタック・フレーム内)：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
フォーマット				0	0	ベクタ・オフセット									

フォーマット/オフセット・ワードのフォーマット・フィールド:

スタック・フレーム・サイズ(フォーマット/オフセット・ワードを含む)を示すフォーマット・コードを保持する。

0000－ショート・フォーマットで、4ワードを取り除く。スタック・フレームからステータス・レジスタとプログラム・カウンタをロードする。

0001－スローアウェイ(throwaway)フォーマットで、4ワードを取り除く。スタック・フレームからステータス・レジスタをロードし、アクティブ・システム・スタックに切り換える。アクティブ・システム・スタックを使用して命令の実行を続行する。

0010－命令エラー・フォーマットで、スタックから6ワードを取り除く。スタック・フレームからステータス・レジスタとプログラム・カウンタをロードし、他のワードは捨てる。

1000－MC68010用ロング・フォーマット。MC68030はこのコードの場合は、フォーマット・エラー例外処理を行なう。

1001－コプロセッサ“命令実行途中の例外”用フォーマットで、10ワードを取り除く。コプロセッサ命令の実行を再開する。

1010－MC68030 用ショート・フォーマットで、16 ワードを取り除き命令の実行を再開する。

1011－MC68030 用ロング・フォーマットで、46 ワードを取り除き命令の実行を再開する。

上記以外のコードの場合、プロセッサはフォーマット・エラー例外処理を実行する。

RTR

Return and Restore Condition Codes・リターンおよびコンディション・コードのリストア

操作：　(SP)→CCR　；SP＋2→SP；
(SP)→PC；SP＋4→SP

アセンブラ・シンタックス：　RTR

属性：　サイズなし

説明：　スタックからコンディション・コードとプログラム・カウンタをプルします。以前のコンディション・コードおよびプログラム・カウンタ値は失われます。ステータス・レジスタのスーパバイザ部分は影響を受けません。

コンディション・コード：　スタックからのコンディション・コードをセット。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	1	1

RTS

Return from Subroutine・サブルーチンからのリターン

操作：　(SP)→PC；SP＋4→SP

アセンブラ・シンタックス：　RTS

属性：　サイズなし

説明：　スタックからプログラム・カウンタ値をプルします。以前のプログラム・カウンタ値は失われます。

コンディション・コード：　影響を受けない。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	0	1

SBCD

Subtract Decimal with Extend・拡張付き10進減算

操作：　デスティネーション$_{10}$－ソース$_{10}$－X→デスティネーション

アセンブラ・シンタックス：　SBCD　Dx, Dy
SBCD－（Ax), －（Ay)

属性：　サイズ＝(バイト)

説明：　デスティネーション・オペランドからソース・オペランドを拡張ビットとともに減算し、結果をデスティネーション・ロケーションに格納します。減算は、2進化10進算術演算を用いて行なわれます。オペランドはパックBCD数値です。この命令には、次の2種類のモードがあります。

1. データ・レジスタとデータ・レジスタ：オペランドは命令で指定するデータ・レジスタにあります。
2. メモリとメモリ：オペランドは命令で指定するアドレス・レジスタを用いたプリデクリメント・アドレッシング・モードで指定します。

この操作はバイト操作に限定されます。

コンディション・コード：

X	N	Z	V	C
*	U	*	U	*

X－キャリ・ビットと同じ。
N－不定
Z－結果が0でなければクリア、それ以外のときは変化しない。
V－不定
C－ボロー(10進)が発生したらセット、それ以外のときはクリア。

注：通常コンディション・コードのZビットは演算を実行する前にプログラムでセットされます。このビットにより、多倍精度演算を終了したときに、演算結果がゼロであるかどうかテストすることができます。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	レジスタRy			1	0	0	0	0	R/M	レジスタRx		

命令フィールド：

レジスタDy/Ayフィールド――デスティネーションのレジスタを指定。
　R/M＝0　　データ・レジスタを指定。
　R/M＝1　　プリデクリメント・アドレッシング・モードで使用するアドレス・レジスタを指定。
R/Mフィールド――オペランドのアドレッシング・モードを指定。
　0－操作はデータ・レジスタとデータ・レジスタ
　1－操作はメモリとメモリ

レジスタDx/Axフィールド――ソース・レジスタを指定。

R/M＝0　データ・レジスタを指定。

R/M＝1　プリデクリメント・アドレッシング・モードで使用するアドレス・レジスタを指定。

Scc

Set According to Condition・条件によるセット

操作：　If　条件が真
then　1s→デスティネーション
else　0s→デスティネーション

アセンブラ・シンタックス：　Scc <ea>

属性：　サイズ＝(バイト)

説明：　指定したコンディション・コードをテストして、条件が真なら実効アドレスで指定されるバイトを真(すべて1)にセットし、そうでなければ偽(すべて0)にセットします。コンディション・コードccは、次の条件の1つを指定します。

CC	キャリ・クリア	0100	$\overline{C}$	LS	ローか同じ	0011	$C+Z$
CS	キャリ・セット	0101	C	LT	より小さい	1101	$N \cdot \overline{V}+\overline{N} \cdot V$
EQ	等しい	0111	Z	MI	マイナス	1011	N
F	真でない	0001	0	NE	等しくない	0110	$\overline{Z}$
GE	大きいか等しい	1100	$N \cdot V+\overline{N} \cdot \overline{V}$	PL	プラス	1010	$\overline{N}1$
GT	より大きい	1110	$N \cdot V \cdot \overline{Z}+\overline{N} \cdot \overline{V} \cdot \overline{Z}$	T	常に真	0000	$\overline{V}$
HI	ハイ	0010	$\overline{C} \cdot \overline{Z}$	VC	ｵｰﾊﾞﾌﾛｰ・ｸﾘｱ	1000	V
LE	小さいか等しい	1111	$Z+N \cdot \overline{V}+\overline{N} \cdot V$	VS	ｵｰﾊﾞﾌﾛｰ・ｾｯﾄ	1001	

コンディション・コード：　影響を受けない。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	コンディション				1	1	実効アドレス					
										モード			レジスタ		

命令フィールド：

コンディション・フィールド――上記の条件の1つに対する2進コードを指定。

実効アドレス・フィールド――真/偽バイトを格納するロケーションを指定。次に示すとおり、データ可変アドレッシング・モードのみ可。

ｱﾄﾞﾚｯｼﾝｸﾞ･ﾓｰﾄﾞ	モード	レジスタ
Dn	000	reg. number : Dn
An	−	−
(An)	010	reg. number : An
(An)＋	011	reg. number : An
−(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

ｱﾄﾞﾚｯｼﾝｸﾞ･ﾓｰﾄﾞ	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	−	−
(d_{16},PC)	−	−
(d_8,PC,Xn)	−	−
(bd,PC,Xn)	−	−
([bd,PC,Xn],od)	−	−
([bd,PC],Xn,od)	−	−

注：Scc命令に続いて同じ実効アドレスをもつNEG.B命令を使用して、Sccの結果をTRUEまたはFALSEからそれと等価の算術値(TRUE＝1、FALSE＝0)に変更することができます。

STOP

Load Status Register and Stop (Privileged instruction)・ステータス・レジスタへのロード、およびストップ(特権命令)

操作：　スーパバイザ状態ではイミディエイト・データ→SR；STOP
ユーザ状態ではTRAP

アセンブラ・シンタックス：　STOP #<data>

属性：　サイズなし

説明：　イミディエイト・オペランドをステータス・レジスタ(ユーザおよびスーパバイザの両部分)に転送し、プログラム・カウンタが次の命令を指すようにインクリメントして、命令のフェッチと実行を停止します。トレース、割込みまたはリセット例外が発生すると、プロセッサは命令の実行を再開します。STOP命令の実行開始時点で命令トレースがイネーブル状態(T0＝1、T1＝0)になっていた場合は、トレース例外が発生します。新しいステータス・レジスタ値によって設定された割込みレベルよりも優先度の高い割込み要求がアサートされると、割込み例外が発生します。それ以外の場合、割込み要求は無視されます。外部リセットがあると常にリセット例外処理が開始されます。

コンディション・コード：　イミディエイト・オペランドに従ってセット。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	1	0
イミディエイト・データ															

命令フィールド：

イミディエイト・フィールド——ステータス・レジスタにロードするデータを指定。

SUB

Subtract・減算

操作：　デスティネーション－ソース→デスティネーション

アセンブラ・シンタックス：　SUB < ea >, Dn

SUB Dn, < ea >

属性：　サイズ＝(バイト、ワード、ロング・ワード)

説明：　デスティネーション・オペランドからソース・オペランドを減算し、結果をデスティネーションに格納します。操作サイズは、バイト、ワード、ロング・ワードが指定できます。命令のモードで、どのオペランドがソースおよびデスティネーションになるかということとそのサイズを指定します。

コンディション・コード：

X	N	Z	V	C
*	*	*	*	*

X－キャリ・ビットと同じ。

N－結果が負であればセット、それ以外のときはクリア。

Z－結果が0であればセット、それ以外のときはクリア。

V－オーバフローが発生すればセット、それ以外のときはクリア。

C－ボローが発生すればセット、それ以外のときはクリア。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
										実効アドレス					
1	0	0	1	レジスタ			Opモード			モード			レジスタ		

命令フィールド：

レジスタ・フィールド――8つのデータ・レジスタのいずれかを指定。

Opモード・フィールド：

バイト	ワード	ロング・ワード	操　作
000	001	010	(< Dn >) － (< ea >) →< Dn >
100	101	110	(< ea >) － (< Dn >) →< ea >

実効アドレス・フィールド——アドレッシング・モードを指定。指定されたロケーションがソース・オペランドの場合は次に示すとおり、すべてのアドレッシング・モードが可。

ｱﾄﾞﾚｯｼﾝｸﾞ・ﾓｰﾄﾞ	モード	レジスタ
Dn	000	reg. number : Dn
An*	001	reg. number : An
(An)	010	reg. number : An
(An)＋	011	reg. number : An
－(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

＊操作サイズがバイトの場合は、アドレス・レジスタ直接は不可。

ｱﾄﾞﾚｯｼﾝｸﾞ・ﾓｰﾄﾞ	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

指定されたロケーションがデスティネーション・オペランドの場合は、次に示すとおり可変メモリ・アドレッシング・モードのみ可。

ｱﾄﾞﾚｯｼﾝｸﾞ・ﾓｰﾄﾞ	モード	レジスタ
Dn	－	－
An	－	－
(An)	010	reg. number : An
(An)＋	011	reg. number : An
－(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

ｱﾄﾞﾚｯｼﾝｸﾞ・ﾓｰﾄﾞ	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	－	－
(d_{16},PC)	－	－
(d_8,PC,Xn)	－	－
(bd,PC,Xn)	－	－
([bd,PC,Xn],od)	－	－
([bd,PC],Xn,od)	－	－

注：1. デスティネーションがデータ・レジスタの場合は、デスティネーション＜ea＞アドレスではなく、デスティネーションDnアドレスを使用します。

2. ほとんどのアセンブラは、デスティネーションがアドレス・レジスタのときはSUBAを使用し、ソースがイミディエイト・データのときには、SUBIまたはSUBQを使用します。

第3章　命令セット

SUBA

Subtract Address・アドレス減算

操作：　デスティネーション－ソース→デスティネーション

アセンブラ・シンタックス：　SUBA ＜ea＞, An

属性：　サイズ＝(ワード、ロング・ワード)

説明：　デスティネーション・アドレス・レジスタからソース・オペランドを減算し、結果をアドレス・レジスタに格納します。操作サイズはワードまたはロング・ワードが指定できます。ワード・サイズ・ソース・オペランドは、操作実行前に32ビットに符号拡張されます。

コンディション・コード：　影響を受けない。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
										実効アドレス					
1	0	0	1	レジスタ			Opモード			モード			レジスタ		

命令フィールド：

レジスタ・フィールド――8つのアドレス・レジスタのいずれかを指定。このフィールドは常にデスティネーションとなる。

Opモード・フィールド――操作サイズを指定。

011－ワード操作、ソース・オペランドがロング・ワードに符号が拡張され、アドレス・レジスタの全32ビットを用いて演算を実行。

111－ロング・ワード操作

実効アドレス・フィールド――ソース・オペランドを指定。次に示すとおり、すべてのアドレッシング・モードが可。

ｱﾄﾞﾚｯｼﾝｸﾞ･ﾓｰﾄﾞ	モード	レジスタ
Dn	000	reg. number : Dn
An	001	reg. number : An
(An)	010	reg. number : An
(An)＋	011	reg. number : An
－(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

ｱﾄﾞﾚｯｼﾝｸﾞ･ﾓｰﾄﾞ	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

SUBI

Subtract Immediate SUBI・イミディエイト減算

操作：　デスティネーション－イミディエイト・データ→デスティネーション

アセンブラ・シンタックス：　SUBI #<data>, <ea>

属性：　サイズ＝(バイト、ワード、ロング・ワード)

説明：　デスティネーション・オペランドからイミディエイト・データを減算し、結果をデスティネーションに格納します。操作サイズは、バイト、ワード、ロング・ワードが指定できます。イミディエイト・データのサイズは操作サイズと同じです。

コンディション・コード：

X	N	Z	V	C
*	*	*	*	*

X－キャリ・ビットの値をセット。

N－結果が負であればセット、それ以外のときはクリア。

Z－結果が0であればセット、それ以外のときはクリア。

V－オーバフローが発生すればセット、それ以外のときはクリア。

C－ボローが発生すればセット、それ以外のときはクリア。

命令フォーマット：

<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">1</td><td rowspan="2">0</td><td rowspan="2">0</td><td colspan="2" rowspan="2">サイズ</td><td colspan="6">実効アドレス</td></tr>
<tr><td colspan="3">モード</td><td colspan="3">レジスタ</td></tr>
<tr><td colspan="8">ワード・データ（16ビット）</td><td colspan="8">バイト・データ（8ビット）</td></tr>
<tr><td colspan="16">ロング・データ（32ビット）</td></tr>
</table>

命令フィールド：

サイズ・フィールド──操作サイズを指定。

00－バイト操作

01－ワード操作

10－ロング・ワード操作

実効アドレス・フィールド――デスティネーション・オペランドを指定。次に示すとおり、データ可変アドレッシング・モードのみ可。

ｱﾄﾞﾚｯｼﾝｸﾞ･ﾓｰﾄﾞ	モード	レジスタ
Dn	000	reg. number : Dn
An	–	–
(An)	010	reg. number : An
(An)+	011	reg. number : An
–(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

ｱﾄﾞﾚｯｼﾝｸﾞ･ﾓｰﾄﾞ	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#<data>	–	–
(d_{16},PC)	–	–
(d_8,PC,Xn)	–	–
(bd,PC,Xn)	–	–
([bd,PC,Xn],od)	–	–
([bd,PC],Xn,od)	–	–

イミディエイト・フィールド――（命令直後のデータ ）

サイズ＝00　データはイミディエイト・ワードの下位バイト。

サイズ＝01　データはイミディエイト・ワード全体。

サイズ＝10　データは次の２つのイミディエイト・ワード。

SUBQ

Subtract Quick・クイック減算

操作：　デスティネーション－イミディエイト・データ→デスティネーション

アセンブラ・シンタックス：　SUBQ　#<data>, <ea>

属性：　サイズ＝(バイト、ワード、ロング・ワード)

説明：　デスティネーション・オペランドからイミディエイト・データ(1-8)を減算します。操作サイズはバイト、ワード、ロング・ワードが指定できます。アドレス・レジスタでは、ワードおよびロング・ワードの操作ができ、この場合、コンディション・コードは影響を受けません。アドレス・レジスタからの減算では、操作サイズに関係なく、デスティネーション・アドレス・レジスタ全体(32ビット)が使用されます。

コンディション・コード：

X	N	Z	V	C
*	*	*	*	*

X－キャリ・ビットの値がセット。
N－結果が負であればセット、それ以外のときはクリア。
Z－結果が0であればセット、それ以外のときはクリア。
V－オーバフローが発生すればセット、それ以外のときはクリア。
C－ボローが発生すればセット、それ以外のときはクリア。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	データ			1	サイズ		実効アドレス					
										モード			レジスタ		

命令フィールド：

データ・フィールド──3ビットのイミディエイト・データで、1-7は1-7を0は8を表わす。
サイズ・フィールド──操作サイズを指定。
　00－バイト操作
　01－ワード操作
　10－ロング・ワード操作

実効アドレス・フィールド——デスティネーションのロケーションを指定。次に示すとおり、可変アドレッシング・モードのみ可。

アドレッシング・モード	モード	レジスタ
Dn	000	reg. number : Dn
An*	001	reg. number : An
(An)	010	reg. number : An
(An)+	011	reg. number : An
−(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

アドレッシング・モード	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#<data>	−	−
(d_{16},PC)	−	−
(d_8,PC,Xn)	−	−
(bd,PC,Xn)	−	−
([bd,PC,Xn],od)	−	−
([bd,PC],Xn,od)	−	−

＊ワードおよびロング・ワードのみ。

SUBX

Subtract with Extend・拡張付き減算

操作：　デスティネーション－ソース－X→デスティネーション

アセンブラ・シンタックス：　SUBX Dx, Dy
SUBX －(Ax), －(Ay)

属性：　サイズ＝(バイト、ワード、ロング・ワード)

説明：　デスティネーション・オペランドからソース・オペランドを拡張ビットとともに減算し、結果をデスティネーションに格納します。この命令には、次の2種類のモードがあります。

1. データ・レジスタとデータ・レジスタ：オペランドは命令で指定するデータ・レジスタにあります。
2. メモリとメモリ：オペランドは命令で指定するアドレス・レジスタを用いたプリデクリメント・アドレッシング・モードで指定します。

オペランドのサイズは、バイト、ワード、またはロング・ワードで指定されます。

コンディション・コード：

X	N	Z	V	C
*	*	*	*	*

X－キャリ・ビットの値をセット。
N－結果が負であればセット、それ以外のときはクリア。
Z－結果が0であればセット、それ以外のときはクリア。
V－オーバフローが発生すればセット、それ以外のときはクリア。
C－キャリが発生すればセット、それ以外のときはクリア。

注：通常コンディション・コードのZビットは演算を実行する前にプログラムでセットされます。このビットにより、多倍精度演算を終了したときに、演算結果がゼロであるかどうかテストすることができます。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	レジスタRy			1	サイズ		0	0	R/M	レジスタRx		

命令フィールド：

レジスタDy/Ayフィールド――デスティネーションのレジスタを指定。

R/M＝0　データ・レジスタを指定。

R/M＝1　プリデクリメント・アドレッシング・モードで使用するアドレス・レジスタを指定。

サイズ・フィールド――操作サイズを指定。

00－バイト操作

01－ワード操作

10－ロング・ワード操作

R/Mフィールド――オペランドのアドレッシング・モードを指定。

0－データ・レジスタとデータ・レジスタの操作

1－メモリとメモリの操作

レジスタDx/Axフィールド――ソース・レジスタを指定。

R/M＝0　データ・レジスタを指定。

R/M＝1　プリデクリメント・アドレッシング・モードで使用するアドレス・レジスタを指定。

SWAP

Swap Register Halves • レジスタ半分交換

操作：　レジスタ［31：16］←→レジスタ［15：0］

アセンブラ・シンタックス：　SWAP Dn

属性：　サイズ＝(ワード)

説明：　データ・レジスタの上位ワードと下位ワード(各16ビット)の内容を入れ替えます。

コンディション・コード：

X	N	Z	V	C
—	*	*	0	0

X－影響を受けない。

N－32ビットの結果の最上位ビットがセットされればセット、それ以外のときはクリア。

Z－32ビットの結果が0であればセット、それ以外のときはクリア。

V－常にクリア。

C－常にクリア。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	1	0	0	0	レジスタ		

命令フィールド：

レジスタ・フィールド——交換するデータ・レジスタを指定。

TAS

Test and Set an Operand・ オペランドのテストとセット

操作：　デスティネーションのテスト結果→コンディション・コード；1→デスティネーションのビット7

アセンブラ・シンタックス：　TAS ＜ea＞

属性：　サイズ＝(バイト)

説明：　実効アドレス・フィールドでアドレス指定されたバイト・オペランドのテストおよびセットを行ないます。オペランドの現在の値をテストし、コンディション・コードのNとZをセットします。また、オペランドの最上位ビットもセットします。操作は、中断なしで操作が完了するリード-モディファイ-ライト・メモリ・サイクルを使用します。この命令は複数のプロセッサの同期をとるために、フラグおよびセマフォの使用をサポートしています。

コンディション・コード：

X	N	Z	V	C
—	*	*	0	0

X－影響を受けない。

N－オペランドの最上位がセットされればセット、それ以外のときはクリア。

Z－オペランドが0であればセット、それ以外のときはクリア。

V－常にクリア。

C－常にクリア。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	0	1	1	実効アドレス					
										モード			レジスタ		

命令フィールド：

実効アドレス・フィールド――テストするオペランドを指定。次に示すとおり、データ可変アドレッシング・モードのみ可。

アドレッシング・モード	モード	レジスタ
Dn	000	reg. number : Dn
An	–	–
(An)	010	reg. number : An
(An)＋	011	reg. number : An
－(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

アドレッシング・モード	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	–	–
(d_{16},PC)	–	–
(d_8,PC,Xn)	–	–
(bd,PC,Xn)	–	–
([bd,PC,Xn],od)	–	–
([bd,PC],Xn,od)	–	–

TRAP

Trap・トラップ

操作：　SSP －2→SSP；フォーマット/オフセット→（SSP）；
SSP －4→SSP；PC→（SSP）；SSP－2→SSP：
SR→（SSP）；ベクタ・アドレス→PC

アセンブラ・シンタックス：　TRAP #<vector>

属性：　サイズなし

説明：　TRAP #<vector>例外を発生します。この命令は、命令のイミディエイト・オペランド（ベクタ）に32を加えてベクタ番号を生成します。ベクタ値の範囲は0～15で、これにより16個のベクタを提供します。

コンディション・コード：　影響を受けない。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	0	ベクタ			

命令フィールド：

ベクタ・フィールド――トラップのベクタ番号を指定。

TRAPcc

Trap on Condition • 条件トラップ

操作：　条件が真であればTRAP

アセンブラ・シンタックス：　TRAPcc
TRAPcc.W　#<data>
TRAPcc.L　#<data>

属性：　サイズなし。またはサイズ＝(ワード、ロング・ワード)

説明：　指定された条件が真であれば、TRAPcc例外が発生します。ベクタ番号は7です。プロセッサは次の命令ワード(現在プログラム・カウンタに入っている値)のアドレスをスタックにプッシュします。条件が真でない場合、プロセッサは何も行なわず、次の命令の実行に移ります。オペレーション・ワードの次にイミディエイト・データ・オペランドを置いておきます。このオペランドは、トラップ・ハンドラが使用することができます。コンディション・コードccは、次の条件の1つを指定します。

CC	キャリ・クリア	0100	$\overline{C}$	LS	ローか同じ	0011	$C+Z$
CS	キャリ・セット	0101	C	LT	より小さい	1101	$N \cdot \overline{V} + \overline{N} \cdot V$
EQ	等しい	0111	Z	MI	マイナス	1011	N
F	真でない	0001	0	NE	等しくない	0110	$\overline{Z}$
GE	大きいか等しい	1100	$N \cdot V + \overline{N} \cdot \overline{V}$	PL	プラス	1010	$\overline{N}$
GT	より大きい	1110	$N \cdot V \cdot \overline{Z} + \overline{N} \cdot \overline{V} \cdot \overline{Z}$	T	常に真	0000	1
HI	ハイ	0010	$\overline{C} \cdot \overline{Z}$	VC	ｵｰﾊﾞﾌﾛｰ•ｸﾘｱ	1000	$\overline{V}$
LE	小さいか等しい	1111	$Z + N \cdot \overline{V} + \overline{N} \cdot V$	VS	ｵｰﾊﾞﾌﾛｰ•ｾｯﾄ	1001	V

コンディション・コード：　影響を受けない。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	コンディション				1	1	1	1	1	Opモード		
オプションのワード															
またはロング・ワード															

命令フィールド：

コンディション・フィールド――上記条件の1つに対する2進コード。

Opモード・フィールド――命令のフォームを選択。
010 －ワード・サイズのオペランドが続く命令。
011 －ロング・ワード・サイズのオペランドが続く命令。
100 －オペランドのない命令。

TRAPV

***Trap on Overflow*・オーバフロー・トラップ**

操作：　V＝1であればTRAP

アセンブラ・シンタックス：　TRAPV

属性：　サイズなし

説明：　オーバフロー・フラグがセットされていれば、TRAPV例外(ベクタ番号7)が発生します。オーバフロー・フラグがセットされていなければ、プロセッサは何もせず、プログラムの実行は次の命令から続行されます。

コンディション・コード：　影響を受けない。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	1	0

TST

Test an Operand • オペランドのテスト

操作：　デスティネーションのテスト結果→コンディション・コード

アセンブラ・シンタックス：　TST <ea>

属性：　サイズ＝(バイト、ワード、ロング・ワード)

説明：　オペランドを0と比較し、テスト結果に応じてコンディション・コードをセットします。操作サイズはビット、ワード、ロング・ワードが指定できます。

コンディション・コード：

X	N	Z	V	C
—	*	*	0	0

X－影響を受けない。

N－オペランドが負であればセット、それ以外のときはクリア。

Z－オペランドが0であればセット、それ以外のときはクリア。

V－常にクリア。

C－常にクリア。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
										実効アドレス					
										モード			レジスタ		
0	1	0	0	1	0	1	0	サイズ							

命令フィールド：

サイズ・フィールド──操作サイズを指定。

00－バイト操作

01－ワード操作

10－ロング・ワード操作

実効アドレス・フィールド──デスティネーション・オペランドを指定。操作サイズがワードまたはロング・ワードの場合は、全アドレッシング・モードが可。操作サイズがバイトの場合は、次に示すとおり、データ・アドレッシング・モードのみ可。

ｱﾄﾞﾚｯｼﾝｸﾞ･ﾓｰﾄﾞ	モード	レジスタ
Dn	000	reg. number : Dn
An	−	−
(An)	010	reg. number : An
(An)+	011	reg. number : An
−(An)	100	reg. number : An
(d_{16},An)	101	reg. number : An
(d_8,An,Xn)	110	reg. number : An
(bd,An,Xn)	110	reg. number : An
([bd,An,Xn],od)	110	reg. number : An
([bd,An],Xn,od)	110	reg. number : An

ｱﾄﾞﾚｯｼﾝｸﾞ･ﾓｰﾄﾞ	モード	レジスタ
(xxx).W	111	000
(xxx).L	111	001
#〈data〉	−	−
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

UNLK

Unlink・リンク解除

操作：　An → SP；(SP) → An；SP + 4 → SP

アセンブラ・シンタックス：　UNLK An

属性：　サイズなし

説明：　指定されたアドレス・レジスタの内容をスタック・ポインタにロードし、ついでアドレス・レジスタにスタックの先頭から取り出したロング・ワードをロードします。

コンディション・コード：　影響を受けない。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	1	1	レジスタ		

命令フィールド：

レジスタ・フィールド――命令のアドレス・レジスタを指定。

UNPK

Unpack BCD・アンパックBCD

操作：　ソース(パックBCD)＋調整値→デスティネーション(アンパックBCD)

アセンブラ・シンタックス：　UNPACK −(Ax), −(Ay), #＜調整値＞
UNPK Dx, Dy, #＜調整値＞

属性：　サイズなし

説明：　ソース・オペランドの2つのBCDディジットを2バイトの下位ニブルに置き、両バイトの上位ニブルに0を置きます。次にこのアンパック値に調整値を加算します。コンディション・コードは変化しません。

両方のオペランドがデータ・レジスタのときには、命令はソース・レジスタの内容をアンパックし、拡張ワードを加算して、結果をデスティネーション・レジスタに置きます。デスティネーション・レジスタの上位ワードは影響を受けません。

ソース (Dx)：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u	u	u	u	u	u	u	u	a	b	c	d	e	f	g	h

中間の拡張：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	a	b	c	d	0	0	0	0	e	f	g	h

調整値ワードの加算：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
16ビット拡張															

デスティネーション(Dy)：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
v	v	v	v	a'	b'	c'	d'	w	w	w	w	e'	f'	g'	h'

指定されたアドレッシング・モードがプリデクリメントの場合、命令はソース・アドレスにあるバイトから2つのBCD ディジットを抽出します。このディジットをアンパックし、調整ワードを加算したのち、デスティネーション・アドレスに2バイトを書き込みます。

ソース(Ax)：

7	6	5	4	3	2	1	0
a	b	c	d	e	f	g	h

中間の拡張：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	a	b	c	d	0	0	0	0	e	f	g	h

調整ワードの加算：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
16ビット拡張															

デスティネーション（Ay）：

7	6	5	4	3	2	1	0
v	v	v	v	a'	b'	c'	d'
w	w	w	w	e'	f'	g'	h'

コンディション・コード： 影響を受けない。

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	レジスタDy/Ay			1	1	0	0	0	R/M	レジスタDx/Ax		
16ビット拡張：調整															

命令フィールド：

レジスタDy/Ayフィールド――デスティネーションのレジスタを指定。

R/M＝0　データ・レジスタを指定。

R/M＝1　プリデクリメント・アドレッシング・モードで使用するアドレス・レジスタを指定。

R/Mフィールド――オペランドのアドレッシング・モードを指定。

0－操作はデータ・レジスタとデータ・レジスタ

1－操作はメモリとメモリ

レジスタDx/Axフィールド――データ・レジスタを指定。

R/M＝0　データ・レジスタを指定。

R/M＝1　プリデクリメント・アドレッシング・モードで使用するアドレス・レジスタを指定。

調整フィールド――ソース・オペランドに加算されるイミディエイト・データ・ワード。適当な定数を調整値として使用して、BCDから希望のコードに変換する。ASCIIに使用する定数は＄3030、EBCDICには＄F0F0を使用する。

3.4　CASおよびCAS2命令の使用法

CAS命令はメモリ・ロケーションの値とデータ・レジスタの値を比較し、それらの値が等しければ、2番目のデータ・レジスタの値をメモリ・ロケーションにコピーします。これにより、システム・カウンタ、履歴情報、およびグローバル共有ポインタを更新することができます。命令は不可分リード・モディファイ・サイクルを使用しており、CASがメモリ・ロケーションを読み出した後は、CASが新しい値を書き込むまで、どの命令もそのロケーションを変更してはなりません。これはシングル・プロセッサ・システム、マルチタスキング環境、およびマルチプロセッサ環境において安全保護を与えます。シングル・プロセッサ・システムでは、操作は割込みルーチンの命令から保護されています。マルチタスキング環境では、ほかのどのタスクもシステム変数に新しい値を書き込むのを妨害することはできません。マルチプロセッサ環境では、ほかのプロセッサはCAS命令が完了しないと、グローバル・ポインタにアクセスすることはできません。

次のコーディング例は、ロケーションSYS_CNTRにあるカウンタで、システムのプロセスまたはプロセッサによる操作の実行回数をカウントするルーチンを示します。このルーチンはレジスタD0にあるカウントの現在の値を取り出して、新しいカウント値をレジスタD1に格納します。CAS命令は新しいカウントが有効であれば、それをSYS_CNTRにコピーします。しかし、別のユーザがカウントが格納されたときからCAS命令のリード・モディファイ・サイクルが発生するまでの間にカウントをインクリメントしていた場合、このルーチンはサイクルの書込み部分でSYS_CNTRの新しいカウントをD0にコピーし、分岐を行なって再実行します。次のコード・シーケンスでは、SYS_CNTRは正しくインクリメントされます。

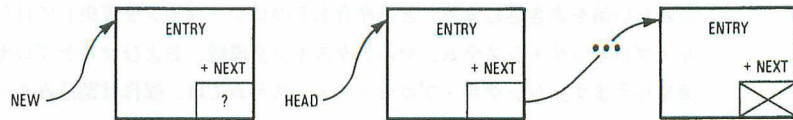
```
         MOVE.W    SYS_CNTR, D0        カウンタの前の値を読む
INC_LOOP MOVE.W    D0, D1              そのコピーを作成する
         ADDQ.W    #1,D1               それをインクリメントする
         CAS.W     D0, D1, SYS_CNTR    カウンタの値が前と同じであれば、それを更新する
         BNE       INC_LOOP            そうでなければ、再試行する
```

CAS命令とCAS2命令を使用して、システムのリンク・リストを安全に操作することができます。例に示すように、HEADという1つのロケーションを制御するだけで、後入れ先出し(last-in-first-out)リンク・リストを管理することができます。リストが空のときは、HEADにはNULLポインタ(0)が入っています。空でないときは、最後にリストに追加された要素のアドレスが入ります。次のコーディング例は、要素を挿入するためのものです。MOVE命令はロケーションHEADのアドレスをD0、および挿入中の要素のNEXTポインタにロードし、新しい要素のアドレスをD1に挿入します。CAS命令は、HEADのアドレスが変更されない場合は、挿入された要素のアドレスをロケーションHEADに格納します。HEADに新しいアドレスが含まれている場合、この命令は新しいアドレスをD0にロードし、2番目のMOVE命令に分岐して、再試行を行ないます。図3-3に挿入操作の様子を示します。

CAS2命令はCAS命令とよく似ていますが、CAS2命令は2つの比較を行なって、比較の結果が等しければ、2つの変数を更新します。両方の比較の結果が等しい場合、CAS2は新しい値をデスティネーション・アドレスにコピーします。いずれかの比較の結果が等しくない場合は、デスティネーション・アドレスの値を比較オペランドにコピーします。

次のコーディング例は、CAS2命令を使用してリンク・リストから1つの要素を削除するというものです。最初のLEA命令はHEADの実効アドレスをA0にロードします。MOVE命令はポインタHEADのアドレスをD0にロードします。TST命令はリストが空かどうかをチェックし、BEQ命令

```
SINSERT                                                新しいエントリをA1にあるアドレスに割り当てる。
          MOVE. L   HEAD, D0                           ヘッド・ポインタ値をD0へ転送する。
SILDOOP   MOVE. L   D0, (NEXT, A1)                     新しいエントリで順方向リンクを確立する。
          MOVE. L   A1, D1                             新しいエントリ・ポインタ値をD1へ転送する。
          CAS. L    D0, D1, HEAD                       依然としてスタックの先頭を指している場合は、ヘッド・ポインタを
                                                       更新する。
          BNE       SILOOP                             そうでない場合は、再度実行する。
```

要素を挿入する前

要素を挿入した後

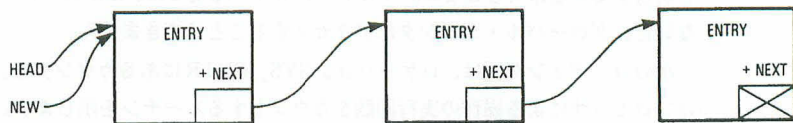



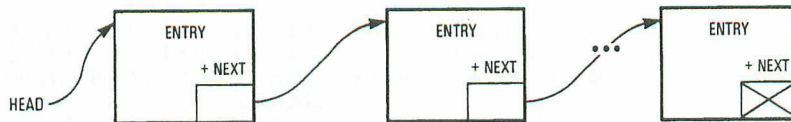
図3-3　リンク・リストの挿入

はリストが空の場合はSDEMPTYというラベルのルーチンに分岐します。リストが空でなかった場合は、2番目のLEA命令がリスト上で最も新しい要素のNEXTポインタのアドレスをA1にロードし、その後のMOVE命令はポインタの内容をD1にロードします。CAS2命令は最新の構造のアドレスをHEADの値と比較し、さらにD1にあるアドレスをA1にあるアドレスのポインタと比較します。このルーチンの実行中に、別のルーチンによってどの要素も挿入または削除されなかったときは、これらの比較の結果は同じであり、CAS命令は新しい値をロケーションHEADに格納します。ある要素が挿入または削除された場合、CAS命令はロケーションHEADにある新しいアドレスをD0にロードし、BNE命令はTST命令に分岐して再試行を行ないます。図3-4にリンク・リストから要素を削除する様子を示します。

CAS2命令を使用して先入れ先出し方式の二重リンク・リストを正しく維持することができます。二重リンク・リストは2つの制御ロケーションLIST__PUTおよびLIST__GETを必要とし、このLIST__PUTおよびLIST__GETにはそれぞれ最後にリストに挿入されたポインタと次に除去するポインタがあります。リストが空の場合は、両方のポインタともNULL(0)です。

次のコーディング例は、二重リンク・リストに1つの要素を挿入する様子を示します。最初の2つの命令は、LIST__PUTとLIST__GETの実効アドレスをそれぞれレジスタA0とA1にロードします。次の命令は新しい要素のアドレスをレジスタD2に転送します。もう1つのMOVE命令は、LIST__PUTのアドレスをレジスタD0に転送します。ラベルDILOOPでは、TST命令がD0にある値をテストし、D0がゼロのときにはBEQ命令がMOVE命令に分岐します。リストが空であると仮定すると、続いてこのMOVE命令が実行されます。この命令はD0のゼロの値を新しい要素のNEXTとLASTポインタに転送します。次にこれらのポインタの両方ともまだゼロであるとすれば、CAS2命令は新しい要素のアドレスをLIST__PUTとLIST__GETの両方に転送します。そうでない場合は、BNE命令がラベルDILOOPにあるTST命令に分岐して再試行を行ないます。このとき、BEQ命令は分岐せず、次のMOVE命令がD0のアドレスを新しい要素のNEXTポインタに転送します。CLR命令はレジスタD1をゼロにクリアし、MOVE命令はゼロを新しい要素のLASTポインタに転送します。LEA命令は、最後に挿入されたLASTポインタのアドレスをA1にロードします。LIST__PUTポインタとA1のポインタが変更されていなければ、CAS命令は新しい要素のアドレスをこれ

```
SDELETE
        LEA      HEAD, A0                  ヘッド・ポインタのアドレスをA0にロードする。
        MOVE. L  (A0), D0                  ヘッド・ポインタの値をD0へ転送する。
SDLOOP  TST. L   D0                        ヘッド・ポインタがヌルかどうかチェックする。
        BEQ      SDEMPTY                   空であれば、何も削除しない。
        LEA      (NEXT, D0), A1            順方向リンクのアドレスをA1にロードする。
        MOVE. L  (A1), D1                  順方向リンクの値をD1に入れる。
        CAS2. L  D0 : D1, D1 : D1,(A0) : (A1)  依然として削除されるべきエントリを指している場合は、ヘッド・ポ
                                           インタと順方向ポインタを更新する。
        BNE      SDLOOP                    そうでない場合は、再度実行する。
SDEMPTY                                    削除が正常に終了し、削除されたエントリのアドレスがD0(ヌルの場
                                           合もある)に入っている。
```

要素を削除する前

要素を削除した後

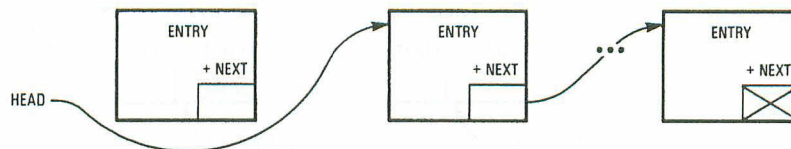



図3-4　リンク・リストの削除

らのポインタに格納します。図3-5に新しい要素を二重リンク・リストに挿入する様子を示します。

二重リンク・リストから1つの要素を削除するためのコーディングも、これとよく似ています。最初の2つの命令がポインタLIST_PUTとLIST_GETの実効アドレスを、それぞれレジスタA0とA1にロードします。

ラベルDDLOOPにあるMOVE命令は、LIST_GETポインタをレジスタD1に転送します。それに続くBEQ命令は、ポインタがゼロのときに分岐してこのルーチンから抜け出します。MOVE命令は削除する要素のLASTポインタをレジスタD2に転送します。これがリストで最後の要素ではないと仮定すれば、コンディション・コードのZビットはセットされず、ラベルDDEMPTYへの分岐は行なわれません。LEA命令はD2のアドレスにある要素のNEXTポインタのアドレスをレジスタA2にロードします。次のCLR命令はレジスタD0をゼロにクリアします。CAS2命令はD1のアドレスをLIST_GETポインタ、およびレジスタA2のアドレスと比較します。ポインタが更新されていなければ、CAS2命令はD2のアドレスをLIST_GETポインタにロードし、ゼロをレジスタA2のアドレスにロードします。

リストに1つの要素しかなければ、このルーチンはゼロ・ポインタ値をD2に転送した後、ラベルDDEMPTYにあるCAS2命令に分岐します。この命令はLIST_PUTとLIST_GETにあるアドレスをチェックして、他のルーチンが別の要素を挿入したり、最後の要素を削除していないかどうか確認します。その後、両方のポインタにゼロを転送し、リストを空にしておきます。図3-6に二重リンク・リストの要素を削除する様子を示します。

3.5　ネストしたサブルーチン・コール

LINK命令はアドレスをスタックにプッシュし、そのアドレスが格納されているスタック・アドレ

```
DINSERT                                               (新しいリスト・エントリを割り当て、アドレスをA2にロードする。)
          LEA       LIST_PUT, A0                      ヘッド・ポインタのアドレスをA0に入れる。
          LEA       LIST_GET, A1                      テイル・ポインタのアドレスをA1に入れる。
          MOVE.L    A2, D2                            新しいエントリ・ポインタをD2にロードする。
          MOVE.L    (A0), D0                          ヘッド・エントリのポインタをD0にロードする。
DILOOP    TST. L    D0                                ヘッド・ポインタはヌル（リストにエントリが存在しナイキスト）か。
          BEQ       DIEMPTY                           そうであれば、ポインタを確定するだけでよい。
          MOVE. L   D0, (NEXT, A2)                    ヘッド・ポインタを新しいエントリの順方向ポインタに入れる。
          CLR.L     D1                                ヌル・ポインタの値をD1に入れる。
          MOVE. L   D1, (LAST, A2)                    ヌル・ポインタを新しいエントリの逆方向ポインタに入れる。
          LEA       (LAST, D0), A1                    前のヘッド・エントリの逆方向ポインタをA1にロードする。
          CAS2. L   D0 : D1, D2 : D2, (A0) : (A1)     依然として前のヘッド・エントリを指している場合は、ポインタを更
                                                      新する。
          BNE       DILOOP                            そうでない場合は、再度実行する。
          BRA       DIDONE
DIEMPTY   MOVE. L   D0, (NEXT, A2)                    ヌル・ポインタを新しいエントリの順方向ポインタに入れる。
          MOVE. L   D0, (LAST, A2)                    ヌル・ポインタを新しいエントリの逆方向ポインタに入れる。
          CAS2. L   D0 : D0, D2 : D2, (A0) : (A1)     依然としてエントリがない場合は、両方のポインタをこのエントリに
                                                      設定する。
          BNE       DILOOP                            そうでない場合は、再度実行する。
DIDONE                                                リストへエントリが正常に挿入された。
```

新しいエントリを挿入する前

新しいエントリを挿入した後

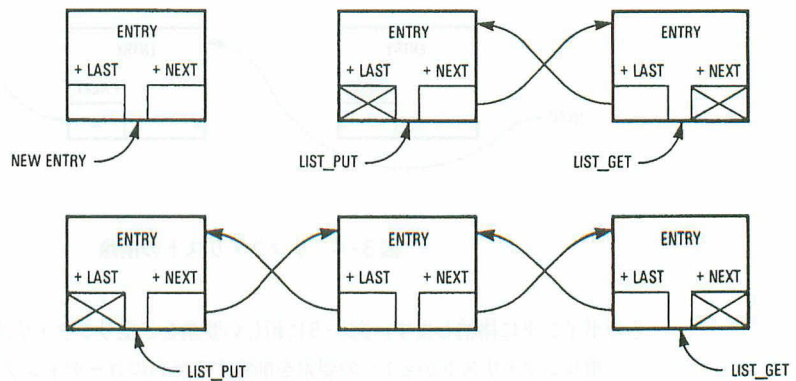



図3-5　二重リンク・リストの挿入

スをセーブし、スタックの領域を確保します。この命令を一連のサブルーチン・コールで使用すれば、スタック・フレームのリンク・リストを作成することができます。

UNLK命令はアドレスをスタック・ポインタにロードし、スタックからそのアドレスにある値をプルすることによって、リストの最後尾からスタック・フレームを除去します。

命令のオペランドがスタック・フレームの最後尾にあるリンク・アドレスのアドレスのときには、結果的にスタックおよびリンク・リストからスタック・フレームが除去されます。

3.6　ビット・フィールド命令

MC68030が扱うデータ・タイプの1つにビット・フィールドがあり、これは最高32ビットの連続ビットで構成されます。ビット・フィールドは実効アドレスからのオフセット値と幅値で定義されます。オフセットは実効アドレスの最上位ビット(ビット7)から-2^{31}～$2^{31}-1$の範囲の値です。幅は1～32の正の数値です。ビット・フィールドの最上位ビットはビット0で、ビット番号は整数のビットの逆方向に付けられています。

この命令セットにはビット・フィールド・オペランドをもつ8つの命令があります。“ビット・フィールドの挿入(BFINS)”命令は、レジスタに格納されたビット・フィールドをビット・フィール

```
DDELETE
          LEA       LIST PUT, A0                   ヘッド・ポインタのアドレスをA0に入れる。
          LEA       LIST GET, A1                   テイル・ポインタのアドレスをA1に入れる。
DDLOOP    MOVE. L   (A1), D1                       テイル・ポインタをD1に転送する。
          BEQ       DDDONE                         リストがなければ抜け出す。
          MOVE. L   (LAST, D1), D2                 逆方向ポインタをD2に入れる。
          BEQ       DDEMPTY                        要素が1つだけしかない場合、ポインタを更新する。
          LEA       (NEXT, D2), A2                 順方向ポインタのアドレスをA2に入れる。
          CLR. L    D0                             ヌル・ポインタの値をD0に入れる。
          CAS2. L   D1 : D1, D2 : D0, (A1) : (A2)  両方のポインタが依然としてこのエントリを指している場合はそれ
                                                   らを更新する。
          BNE       DDLOOP                         そうでない場合は、再度実行する。
          BRA       DDDONE
DDEMPTY   CAS2.L    D1 : D1, D2 : D2, (A1) : (A0)  依然として最初のエントリであれば、ヘッドおよびテイルのポインタ
                                                   をヌルに設定する。
          BNE       DDLOOP                         そうでない場合は、再度実行する。
DDDONE                                             エントリが正常に削除された。削除されたエントリのアドレスがD1
                                                   (ヌルの場合もある)に入っている。
```

新しいエントリを削除する前

新しいエントリを削除した後

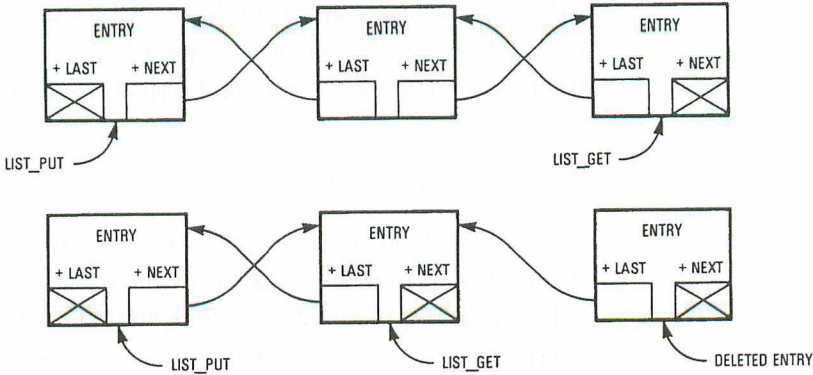



図3-6　二重リンク・リストの削除

ドに挿入します。“符号付きビット・フィールドの抽出(BFEXTS)”命令はビット・フィールドをレジスタの最下位ビット群にロードし、左方向に符号拡張して、レジスタを充てんします。“符号なしビット・フィールドの抽出(BFEXTU)” 命令もビット・フィールドをロードしますが、デスティネーション・レジスタの未使用部分にはゼロが満たされます。

“ビット・フィールドのセット(BFSET)” 命令はフィールドの全ビットを1にセットします。“ビット・フィールドのクリア(BFCLR)” 命令はフィールドをクリアします。また、“ビット・フィールドの変更(BFCHG)” 命令はビット・フィールドの全ビットを補数化します。これらの3つの命令は、すべて対象となるビット・フィールドの前の値をテストし、その結果に従ってコンディション・コードをセットします。“ビット・フィールドのテスト(BFTST)” 命令はフィールドの値をテストし、その結果に従ってコンディション・コードをセットします。ビット・フィールドの内容は変更されません。“ビット・フィールド内の最初の1の検出(BFFFO)” 命令は、ビット・フィールドを、1にセットされているビットが見つかるまでビット0から右に走査し、最初に見つかった1のビット・オフセットを指定されたレジスタにロードします。フィールドでどのビットも1にセットされていなかった場合は、フィールド・オフセットとフィールド幅を加えた値がレジスタにロードされます。

ビット・フィールド命令の重要な用途として、浮動小数点数における指数フィールドの操作があります。IEEEの標準フォーマットでは、最上位ビットは仮数部の符号ビットです。指数値は次の最上位ビット位置から始まります。指数フィールドはバイト境界からは開始されません。この用途には、“ビット・フィールドの抽出(BFEXTU)” 命令および “ビット・フィールドのテスト(BFTST)”

命令が最適ですが、他のビット・フィールド命令を使用することもできます。

周辺デバイスの入力および出力をプログラミングするには、周辺デバイスの制御レジスタのビット・フィールドのテスト、セット、および挿入が必要です。これがビット・フィールド命令のもう1つの用途です。ただし、制御レジスタのロケーションはメモリ・ロケーションではありませんので、常にレジスタ内の他のフィールドに影響を与えずに、レジスタ内のビット・フィールドの挿入や抽出を行なうことができるとはかぎりません。

もう1つ広く使用されているビット・フィールド命令の用途に、ビット・マップ式のグラフィックがあります。これらのメモリ領域では、バイト境界は無視されますので、ビット・フィールド命令で使用するフィールド定義が非常に役立ちます。

3. 7　NOP命令によるパイプラインの同期化

無操作(NOP)命令は具体的な操作は何も行ないませんが、重要な役割を果たします。NOP命令は、保留されているすべてのバス・サイクルが完了するまで待つことにより、命令パイプラインの同期化を図ります。以前の命令はすべてNOPが始まる前に実行を完了します。

3. 8　コンディション・コード

ステータス・レジスタの最下位5ビット(コンディション・コード)は、多数の命令によってセットあるいはクリアされ、命令の結果を示します。この節には、次のリストがあります。

1) すべての命令およびコードのセット方法を掲載したリスト
2) 条件テストのニーモニックとそれらのコンディション・コード・ビットでの意味を一覧にまとめたリスト

コンディション・コードは、次の2つの基準に適合するように開発されました。

●一貫性——命令、使用法、使用状況を通して。

●意味のある結果——有用な情報を提示できる場合以外は変化しない。

命令の一貫性は、すべての命令が一般の命令の特殊ケースに相当する場合でも、同じようにコンディション・コードに影響を与えるという意味です。使用状況の一貫性というのは、ある命令はどのような状況においても、同じようにコンディション・コードに影響を与えるという意味です。また、使用法の一貫性というのは、条件付き命令は、コンディション・コードを同じようにテストし、それが比較、テスト、または転送命令のどれでセットされても同じ結果を与えるという意味です。

3. 8. 1　コンディション・コードの計算

ほとんどの操作は、ソース・オペランドとデスティネーション・オペランドを取り込んで計算し、結果をデスティネーション・ロケーションに格納します。単一オペランド操作はデスティネーション・オペランドを取り込んで、計算結果をそのデスティネーション・ロケーションに格納します。表3-13に、各命令とそれによってコンディション・コードがどのようにセットされるかをまとめて示します。

表3-13　コンディション・コードの計算

操　　作	X	N	Z	V	C	特　　殊　　定　　義
ABCD	*	U	?	U	?	C＝10進キャリ $Z = Z \wedge \overline{Rm} \wedge \ldots \wedge \overline{R0}$
ADD, ADDI, ADDQ	*	*	*	?	?	$V = Sm \wedge Dm \wedge \overline{Rm} \vee \overline{Sm} \wedge \overline{Dm} \wedge Rm$ $C = Sm \wedge Dm \vee \overline{Rm} \wedge Dm \vee Sm \wedge \overline{Rm}$
ADDX	*	*	?	?	?	$V = Sm \wedge Dm \wedge \overline{Rm} \vee \overline{Sm} \wedge \overline{Dm} \wedge Rm$ $C = Sm \wedge Dm \vee \overline{Rm} \wedge Dm \vee Sm \wedge \overline{Rm}$ $Z = Z \wedge \overline{Rm} \wedge \ldots \wedge \overline{R0}$
AND, ANDI, EOR, EORI, MOVEQ, MOVE, OR, ORI, CLR, EXT, NOT, TAS, TST	−	*	*	0	0	
CHK	−	*	U	U	U	
CHK2, CMP2	−	U	?	U	?	Z＝（R＝LB）V（R＝UB） C＝（LB<＝UB）∧（IR<LB）V（R>UB））V（UB<LB）∧（R>UB）∧（R<LB）
SUB, SUBI, SUBQ	*	*	*	?	?	$V = \overline{Sm} \wedge Dm \wedge \overline{Rm} \vee Sm \wedge \overline{Dm} \wedge Rm$ $C = Sm \wedge \overline{Dm} \vee Rm \wedge \overline{Dm} \vee Sm \wedge Rm$
SUBX	*	*	?	?	?	$V = \overline{Sm} \wedge Dm \wedge \overline{Rm} \vee Sm \wedge \overline{Dm} \wedge Rm$ $C = Sm \wedge \overline{Dm} \vee Rm \wedge \overline{Dm} \vee Sm \wedge Rm$ $Z = Z \wedge \overline{Rm} \wedge \ldots \wedge \overline{R0}$
CAS, CAS2, CMP, CMPI, CMPM	−	*	*	?	?	$V = \overline{Sm} \wedge Dm \wedge \overline{Rm} \vee Sm \wedge \overline{Dm} \wedge Rm$ $C = Sm \wedge \overline{Dm} \vee Rm \wedge \overline{Dm} \vee Sm \wedge Rm$
DIVS, DUVI	−	*	*	?	0	V＝除算オーバフロー
MULS, MULU	−	*	*	?	0	V＝乗算オーバフロー
SBCD, NBCD	*	U	?	U	?	C＝10進ボロー $Z = Z \wedge \overline{Rm} \wedge \ldots \wedge \overline{R0}$
NEG	*	*	*	?	?	$V = Dm \wedge Rm$ $C = Dm \vee Rm$
NEGX	*	*	?	?	?	$V = Dm \wedge Rm$ $C = Dm \vee Rm$ $Z = Z \wedge \overline{Rm} \wedge \ldots \wedge \overline{R0}$
BTST, BCHG, BSET, BCLR	−	−	?	−	−	$Z = \overline{Dn}$
BFTST, BFCHG, BFSET, BFCLR	−	?	?	0	0	$N = Dm$ $Z = \overline{Dm} \wedge \overline{Dm-1} \wedge \ldots \wedge \overline{D0}$
BFEXTS, BFEXTU, BFFFO	−	?	?	0	0	$N = Sm$ $Z = \overline{Sm} \wedge \overline{Sm-1} \wedge \ldots \wedge \overline{S0}$
BFINS	−	?	?	0	0	$N = Dm$ $Z = \overline{Dm} \wedge \overline{Dm-1} \wedge \ldots \wedge \overline{D0}$
ASL	*	*	*	?	?	$V = Dm \wedge (\overline{Dm-1} \vee \ldots \vee \overline{Dm-r}) \vee \overline{Dm} \wedge (Dm-1 \vee \ldots + Dm-r)$ $C = \overline{Dm-r+1}$
ASL（R＝0）	−	*	*	0	0	
LSL, ROXL	*	*	*	0	?	C＝Dm−r＋1
LSR（r＝0）	−	*	*	0	0	
ROXL（r＝0）	−	*	*	0	?	C＝X
ROL	−	*	*	0	?	C＝Dm−r＋1
ROL（r＝0）	−	*	*	0	0	
ASR, LSR, ROXR	*	*	*	0	?	C＝Dr−1
ASR, LSR（r＝0）	−	*	*	0	0	
ROXR（r＝0）	−	*	*	0	?	C＝X
ROR	−	*	*	0	?	C＝Dr−1
ROR（r＝0）	−	*	*	0	0	

−＝影響なし
U＝不定、結果は無意味
?＝その他−特殊定義参照
*＝一般の場合
　X＝C
　$N = Rm$
　$Z = \overline{Rm} \wedge \ldots \wedge R0$
Sm＝ソース・オペランド――最上位ビット
Dm＝デスティネーション・オペランド――最上位ビット

Rm＝結果のオペランドの最上位ビット
R＝テストされるレジスタ
n＝ビット番号
r＝シフト回数
LB＝下限
UB＝上限
∧ V＝論理積
V＝論理和
$\overline{Rm}$＝NOT Rm（Rmの否定）

表3-14　条件テスト

ニーモニック	条件	エンコーディング	テスト
T＊	真	0000	1
F＊	偽	0001	0
HI	ハイ	0010	$\overline{C}\cdot\overline{Z}$
LS	ローか同じ	0011	C＋Z
CC (HS)	キャリ・クリア	0100	$\overline{C}$
CS (LO)	キャリ・セット	0101	C
NE	等しくない	0110	$\overline{Z}$
EQ	等しい	0111	Z
VC	オーバフロー・クリア	1000	$\overline{V}$
VS	オーバフロー・セット	1001	V
PL	プラス	1010	$\overline{N}$
MI	マイナス	1011	N
GE	大きいか等しい	1100	$N\cdot V+\overline{N}\cdot\overline{V}$
LT	より小さい	1101	$N\cdot\overline{V}+\overline{N}\cdot V$
GT	より大きい	1110	$N\cdot V\cdot\overline{Z}+\overline{N}\cdot\overline{V}\cdot\overline{Z}$
LE	小さいか等しい	1111	$Z+N\cdot\overline{V}+\overline{N}\cdot V$

・＝論理積
＋＝論理和
$\overline{N}$＝Nの否定
＊ Bcc命令にはありません。

3. 8. 2　条件テスト

表3-14に条件名、エンコーディング、そして条件付き分岐およびセット命令のテストの関係を示します。各条件についてのテストは、コンディション・コード の現在の状態に基づく論理式によって表わされます。この論理式の評価結果が1であれば、条件は「真」になります。論理式の評価結果が0であれば、条件は「偽」になります。たとえば、Tの条件は常に「真」になりますが、EQ条件はコンディション・コードのZビットが「真」のときにだけ「真」になります。

3. 9　命令フォーマットの要約

以下のパラグラフでは、各命令の主要ワードを要約して掲載します。完全な命令は、この主要ワードの次にイミディエイト・データ・フィールド、ディスプレースメント、およびインデックス・オペランドなどのアドレッシング・モード・オペランドが続く形で構成されます。最初の(または、唯一の)第1ワードの最上位4ビットで命令を分類しています。表3-15にこれらのビットの組合せに基づいて命令を分類した、オペレーション・コード(オペコード)マップを示します。

最初のところでは、このオペコード・マップに従って標準的な命令をグループにまとめています。MC68010、MC68020、およびMC68030で追加された命令には、その旨マークが付けてあります。最後の部分に、コプロセッサの命令形式が掲載されています。

表3-15　操作コード・マップ

ビット15〜12	操　　作
0000	ビット操作/MOVEP/イミディエイト
0001	バイト転送
0010	ロング・ワード転送
0011	ワード転送
0100	その他
0101	ADDQ/SUBQ/Scc/DBcc/TRAPcc
0110	Bcc/BSR/BRA
0111	MOVEQ
1000	OR/DIV/SBCD
1001	SUB/SUBX
1010	(未割当て、予約)
1011	CMP/EOR
1100	AND/MUL/ABCD/EXG
1101	ADD/ADDX
1110	シフト/ローテイト/ビット・フィールド
1111	コプロセッサ・インタフェース

MC68030の命令

ORI

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	サイズ		実効アドレス モード			実効アドレス レジスタ		
ワード・データ(16ビット)								バイト・データ(8ビット)							
ロング・データ(32ビット)															

サイズ・フィールド：00＝バイト　01＝ワード　10＝ロング・ワード

ORI to CCR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0
0	0	0	0	0	0	0	0	バイト・データ(8ビット)							

ORI to SR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0
ワード・データ(16ビット)															

CMP2

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	サイズ		0	1	1	実効アドレス モード			実効アドレス レジスタ		
D/A	レジスタ			0	0	0	0	0	0	0	0	0	0	0	0

サイズ・フィールド：00＝バイト　01＝ワード　10＝ロング・ワード

CHK2

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	サイズ		0	1	1	実効アドレス モード			実効アドレス レジスタ		
D/A	レジスタ			1	0	0	0	0	0	0	0	0	0	0	0

サイズ・フィールド：00＝バイト　01＝ワード　10＝ロング・ワード

Bit (Dynamic)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	データ・レジスタ			1	タイプ		実効アドレス モード			実効アドレス レジスタ		

タイプ・フィールド：00＝TST　10＝CLR　01＝CHG　11＝SET

MOVEP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	データ・レジスタ			Opモード			0	0	1	アドレス・レジスタ		
ディスプレースメント(16ビット)															

Opモード・フィールド：100＝メモリからレジスタへのワード転送
101＝メモリからレジスタへのロング・ワード転送
110＝レジスタからメモリへのワード転送
111＝レジスタからメモリへのロング・ワード転送

ANDI

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	サイズ		実効アドレス モード			レジスタ		
ワード・データ(16ビット)								バイト・データ(8ビット)							
ロング・データ(32ビット)															

サイズ・フィールド：00＝バイト　01＝ワード　10＝ロング・ワード

ANDI to CCR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	0	0	1	1	1	1	0	0
0	0	0	0	0	0	0	0	バイト・データ(8ビット)							

ANDI to SR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	0	1	1	1	1	1	0	0
ワード・データ(16ビット)															

SUBI

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	0	サイズ		実効アドレス モード			レジスタ		
ワード・データ(16ビット)								バイト・データ(8ビット)							
ロング・データ(32ビット)															

サイズ・フィールド：00＝バイト　01＝ワード　10＝ロング・ワード

ADDI

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	1	0	サイズ		実効アドレス モード			レジスタ		
ワード・データ(16ビット)								バイト・データ(8ビット)							
ロング・データ(32ビット)															

サイズ・フィールド：00＝バイト　01＝ワード　10＝ロング・ワード

CAS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	サイズ		0	1	1	実効アドレス モード			レジスタ		
0	0	0	0	0	0	0	Du			0	0	0	Dc		

サイズ・フィールド：01＝バイト　10＝ワード　11＝ロング・ワード

CAS2

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	サイズ		0	1	1	1	1	1	1	0	0
D/A1	Rn1			0	0	0	Du1			0	0	0	Dc1		
D/A2	Rn2			0	0	0	Du2			0	0	0	Dc2		

サイズ・フィールド：10＝ワード　11＝ロング・ワード

Bit (Static)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	タイプ		実効アドレス モード			実効アドレス レジスタ		
0	0	0	0	0	0	0	0	ビット番号							

タイプ・フィールド：00＝TST　10＝CLR　01＝CHG　11＝SET

EORI

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	サイズ		実効アドレス モード			実効アドレス レジスタ		
ワード・データ(16ビット)								バイト・データ(8ビット)							
ロング・データ(32ビット)															

サイズ・フィールド：00＝バイト　01＝ワード　10＝ロング・ワード

EORI to CCR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	0	0	1	1	1	1	0	0
0	0	0	0	0	0	0	0	バイト・データ(8ビット)							

EORI to SR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	0	1	1	1	1	1	0	0
ワード・データ(16ビット)															

CMPI

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	0	サイズ		実効アドレス モード			実効アドレス レジスタ		
ワード・データ(16ビット)								バイト・データ(8ビット)							
ロング・データ(32ビット)															

サイズ・フィールド：00＝バイト　01＝ワード　10＝ロング・ワード

MOVES

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	1	0	サイズ		実効アドレス モード			実効アドレス レジスタ		
A/D	レジスタ			dr	0	0	0	0	0	0	0	0	0	0	0

dr フィールド：0＝EAからレジスタ　1＝レジスタからEA

MOVE Byte

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	デスティネーション レジスタ			デスティネーション モード			ソース モード			ソース レジスタ		

レジスタとモードの位置に注意

MOVEA Long

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	デスティネーション レジスタ			0	0	1	ソース					
										モード			レジスタ		

MOVE Long

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	デスティネーション						ソース					
				レジスタ			モード			モード			レジスタ		

レジスタとモードの位置に注意

MOVEA Word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	デスティネーション レジスタ			0	0	1	ソース					
										モード			レジスタ		

MOVE Word

0	0	1	1	デスティネーション						ソース					
				レジスタ			モード			モード			レジスタ		

レジスタとモードの位置に注意

NEGX

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	サイズ		実効アドレス					
										モード			レジスタ		

サイズ・フィールド：00＝バイト　01＝ワード　10＝ロング・ワード

MOVE from SR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	1	1	実効アドレス					
										モード			レジスタ		

CHK

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	レジスタ			サイズ		0	実効アドレス					
										モード			レジスタ		

サイズ・フィールド：10＝ロング・ワード　11＝ワード

LEA

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	レジスタ			1	1	1	実効アドレス					
										モード			レジスタ		

CLR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	1	0	サイズ		実効アドレス モード			レジスタ		

サイズ・フィールド：00＝バイト　01＝ワード　10＝ロング・ワード

MOVE from CCR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	1	0	1	1	実効アドレス モード			レジスタ		

NEG

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	0	0	サイズ		実効アドレス モード			レジスタ		

サイズ・フィールド：00＝バイト　01＝ワード　10＝ロング・ワード

MOVE to CCR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	0	0	1	1	実効アドレス モード			レジスタ		

NOT

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	1	0	サイズ		実効アドレス モード			レジスタ		

サイズ・フィールド：00＝バイト　01＝ワード　10＝ロング・ワード

MOVE to SR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	1	0	1	1	実効アドレス モード			レジスタ		

NBCD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	0	実効アドレス モード			レジスタ		

LINK Long

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	0	0	0	1	レジスタ		
上位ディスプレースメント															
下位ディスプレースメント															

SWAP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	1	0	0	0	レジスタ		

BKPT

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	1	0	0	1	ベクタ		

PEA

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	1	実効アドレス					
										モード			レジスタ		

サイズ・フィールド：00＝バイト　01＝ワード　10＝ロング・ワード

EXT/EXTB

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	Opモード			0	0	0	レジスタ		

Opモード・フィールド：010＝拡張ワード　011＝拡張ロング・ワード　111＝拡張バイト・ロング

MOVEM Registers to EA

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	dr	0	0	1	サイズ	実効アドレス					
										モード			レジスタ		
レジスタ・リスト・マスク															

サイズ・フィールド：0＝ワード転送　1＝ロング・ワード転送

TST

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	0	サイズ		実効アドレス					
										モード			レジスタ		

サイズ・フィールド：00＝バイト　01＝ワード　10＝ロング・ワード

TAS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	0	1	1	実効アドレス					
										モード			レジスタ		

ILLEGAL

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	0	1	1	1	1	1	1	0	0

MULS/MULU Long

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	0	0	実効アドレス モード			実効アドレス レジスタ		
0	レジスタDl			タイプ	サイズ	0	0	0	0	0	0	0	レジスタDh		

タイプ・フィールド：0＝MULU　1＝MULS

サイズ・フィールド：0＝ロング・ワード積　1＝クワッド・ワード積

DIVS/DIVU Long
DIVUL/DIVSL

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	0	1	実効アドレス モード			実効アドレス レジスタ		
0	レジスタDq			タイプ	サイズ	0	0	0	0	0	0	0	レジスタDr		

タイプ・フィールド：0＝DIVU　1＝DIVS

サイズ・フィールド：0＝ロング・ワード被除数　1＝クワッド・ワード被除数

MOVEM EA to Registers

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	dr	0	0	1	サイズ	実効アドレス モード			実効アドレス レジスタ		
レジスタ・リスト・マスク															

サイズ・フィールド：0＝ワード転送　1＝ロング・ワード転送

TRAP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	0	ベクタ			

Link Word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	1	0	レジスタ		
ワード・ディスプレースメント															

UNLK

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	1	1	レジスタ		

MOVE to USP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	0	0	レジスタ		

MOVE from USP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	0	1	レジスタ		

RESET

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	0	0

NOP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	0	1

RTE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	1	1

RTD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	0	0
ディスプレースメント(16ビット)															

RTS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	0	1

TRAPV

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	1	0

RTR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	1	1

MOVEC

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	1	0	1	dr
A/D	レジスタ			制御レジスタ											

dr フィールド：　0＝制御レジスタから汎用レジスタ
　　　　　　　　1＝汎用レジスタから制御レジスタ

制御レジスタ・フィールド：
$ 000 = SFC　　$ 801 = VBR
$ 001 = DFC　　$ 802 = CAAR
$ 002 = CACR　　$ 803 = MSP
$ 800 = USP　　$ 804 = ISP

JSR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	1	0	実効アドレス					
										モード			レジスタ		

JMP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	1	1	実効アドレス					
										モード			レジスタ		

ADDQ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	データ			0	サイズ		実効アドレス					
										モード			レジスタ		

データ・フィールド：3ビットのイミディエイト・データ、1-7は値1-7、0は値8を表わす。
サイズ・フィールド：00＝バイト　01＝ワード　10＝ロング・ワード

Scc

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	コンディション				1	1	実効アドレス					
										モード			レジスタ		

DBcc

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	コンディション				1	1	0	0	1	レジスタ		
ディスプレースメント(16ビット)															

TRAPcc

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	コンディション				1	1	1	1	1	Opモード		
オプションのワード															
オプションのロング・ワード															

Opモード・フィールド：010＝ワード・オペランド　011＝ロング・ワード・オペランド
100＝オペランドなし

SUBQ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	データ			1	サイズ		実効アドレス					
										モード			レジスタ		

データ・フィールド：3ビットのイミディエイト・データ、1-7は値1-7、0は値8を表わす。
サイズ・フィールド：00＝バイト　01＝ワード　10＝ロング・ワード

Bcc

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	コンディション				8ビット・ディスプレースメント							
8ビット・ディスプレースメント＝$00の場合、16ビット・ディスプレースメント															
8ビット・ディスプレースメント＝$FFの場合、32ビット・ディスプレースメント															

BRA

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	0	8ビット・ディスプレースメント							
8ビット・ディスプレースメント＝$00の場合、16ビット・ディスプレースメント															
8ビット・ディスプレースメント＝$FFの場合、32ビット・ディスプレースメント															

BSR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	1	8ビット・ディスプレースメント							
8ビット・ディスプレースメント＝$00の場合、16ビット・ディスプレースメント															
8ビット・ディスプレースメント＝$FFの場合、32ビット・ディスプレースメント															

MOVEQ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	レジスタ			0	データ							

データ・フィールド：データはロング・ワード・オペランドに符号拡張され、全32ビットがデータ・レジスタに転送される。

OR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	レジスタ			Opモード			実効アドレス					
										モード			レジスタ		

Opモード・フィールド：

バイト	ワード	ロング・ワード	操　作
000	001	010	(〈ea〉)v(〈Dn〉)→〈Dn〉
100	101	110	(〈Dn〉)v(〈ea〉)→〈ea〉

DIVS/DIVU Word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	レジスタ			タイプ	1	1	実効アドレス					
										モード			レジスタ		

タイプ・フィールド：0＝DIVU　1＝DIVS

SBCD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	レジスタRy			1	0	0	0	0	R/M	レジスタRx		

R/Mフィールド：0＝データ・レジスタからデータ・レジスタ　1＝メモリからメモリ
R/M＝0 両方のレジスタがデータ・レジスタ
R/M＝1 両方のレジスタがプリデクリメント・アドレッシング・モードで使用するアドレス・レジスタ

PACK

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	レジスタDy/Ay			1	0	1	0	0	R/M	レジスタDx/Ax		
16ビット拡張：調整															

R/Mフィールド：0＝データ・レジスタからデータ・レジスタ　1＝メモリからメモリ
R/M＝0 両方のレジスタがデータ・レジスタ
R/M＝1 両方のレジスタがプリデクリメント・アドレッシング・モードで使用するアドレス・レジスタ

UNPK

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	レジスタ Dy/Ay			1	1	0	0	0	R/M	レジスタ Dx/Ax		
16 ビット拡張：調整															

R/M フィールド：0＝データ・レジスタからデータ・レジスタ　1＝メモリからメモリ
R/M＝0 両方のレジスタがデータ・レジスタ
R/M＝1 両方のレジスタがプリデクリメント・アドレッシング・モードで使用するアドレス・レジスタ

SUB

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
										実効アドレス					
1	0	0	1	レジスタ			Op モード			モード			レジスタ		

Op モード・フィールド：

バイト	ワード	ロング・ワード	操　作
000	001	010	⟨⟨Dn⟩⟩－⟨⟨ea⟩⟩ → ⟨Dn⟩
100	101	110	⟨⟨ea⟩⟩－⟨⟨Dn⟩⟩ → ⟨ea⟩

SUBA

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
										実効アドレス					
1	0	0	1	レジスタ			Op モード			モード			レジスタ		

Op モード・フィールド：

ワード	ロング・ワード	操　作
011	111	⟨⟨An⟩⟩－⟨⟨ea⟩⟩ → ⟨An⟩

SUBX

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	レジスタ Ry			1	サイズ		0	0	R/M	レジスタ Rx		

サイズ・フィールド：00＝バイト　01＝ワード　10＝ロング・ワード
R/M フィールド：0＝データ・レジスタからデータ・レジスタ　1＝メモリからメモリ
R/M＝0 両方のレジスタがデータ・レジスタ
R/M＝1 両方のレジスタがプリデクリメント・アドレッシング・モードで使用するアドレス・レジスタ

CMP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
										実効アドレス					
1	0	1	1	レジスタ			Op モード			モード			レジスタ		

Op モード・フィールド：

バイト	ワード	ロング・ワード	操　作
000	001	010	⟨⟨Dn⟩⟩－⟨⟨ea⟩⟩

CMPA

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
										実効アドレス					
1	0	1	1	レジスタ			Op モード			モード			レジスタ		

Op モード・フィールド：

ワード	ロング・ワード	操　作
011	111	⟨⟨An⟩⟩－⟨⟨ea⟩⟩

EOR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	レジスタ			Opモード			実効アドレス					
										モード			レジスタ		

Opモード・フィールド：

バイト	ワード	ロング・ワード	操　作
100	101	110	((ea)) ⊕ ((Dn)) → (ea)

CMPM

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	レジスタ Ax			1	サイズ		0	0	1	レジスタ Ay		

サイズ・フィールド：00＝バイト　01＝ワード　10＝ロング・ワード

AND

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	レジスタ			Opモード			実効アドレス					
										モード			レジスタ		

Opモード・フィールド：

バイト	ワード	ロング・ワード	操　作
000	001	010	((ea))∧((Dn)) → (Dn)
100	101	110	((Dn))∧((ea)) → (ea)

MULS/MULU Word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	レジスタ			タイプ	1	1	実効アドレス					
										モード			レジスタ		

タイプ・フィールド：0＝MULU　1＝MULS

ABCD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	レジスタ Rx			1	0	0	0	0	R/M	レジスタ Ry		

R/M フィールド：0＝データ・レジスタからデータ・レジスタ　1＝メモリからメモリ

R/M＝0 両方のレジスタがデータ・レジスタ

R/M＝1 両方のレジスタがプリデクリメント・アドレッシング・モードで使用するアドレス・レジスタ

EXG Data Registers

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	レジスタ Dx			1	0	1	0	0	0	レジスタ Dy		

EXG Address Registers

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	レジスタ Ax			1	0	1	0	0	1	レジスタ Ay		

EXG Data Register and Address Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	レジスタ Dx			1	1	0	0	0	1	レジスタ Ay		

ADD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	レジスタ			Opモード			実効アドレス モード			実効アドレス レジスタ		

Opモード・フィールド：

バイト	ワード	ロング・ワード	操作
000	001	010	((ea))+((Dn))→(Dn)
100	101	110	((Dn))+((ea))→(ea)

ADDA

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	レジスタ			Opモード			実効アドレス モード			実効アドレス レジスタ		

Opモード・フィールド：

ワード	ロング・ワード	操作
011	111	((ea))+((An))→(An)

ADDX

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	レジスタRx			1	サイズ		0	0	R/M	レジスタRy		

サイズ・フィールド：00＝バイト　01＝ワード　10＝ロング・ワード
R/M フィールド：0＝データ・レジスタからデータ・レジスタ　1＝メモリからメモリ
R/M＝0 両方のレジスタがデータ・レジスタ
R/M＝1 両方のレジスタがプリデクリメント・アドレッシング・モードで使用するアドレス・レジスタ

Shift/Rotate Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	カウント/レジスタ			dr	サイズ		I/R	タイプ		レジスタ		

カウント・レジスタ・フィールド：
I/Rフィールド＝0、シフト・カウント指定
I/Rフィールド＝1、シフト・カウントを保持するデータ・レジスタを指定
drフィールド：0＝右　1＝左
サイズ・フィールド：00＝バイト　01＝ワード　10＝ロング・ワード
I/Rフィールド：0＝イミディエイト・シフト・カウント　1＝レジスタ・シフト・カウント
タイプ・フィールド：00＝算術シフト　01＝論理シフト　10＝拡張付きローテイト
11＝ローティト

Shift/Rotate Memory

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	タイプ		dr	1	1	実効アドレス モード			実効アドレス レジスタ		

タイプ・フィールド：00＝算術シフト　01＝論理シフト　10＝拡張付きローテイト
11＝ローテイト
drフィールド：0＝右　1＝左

Bit Field

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
										実効アドレス					
1	1	1	0	1	タイプ			1	1	モード			レジスタ		
0	レジスタ			Do	オフセット					Dw	幅				

タイプ・フィールド：000＝BFTST　　100＝BFCLR
001＝BFEXTU　　101＝BFFFO
010＝BFCHG　　110＝BFSET
011＝BFEXTS　　111＝BFINS

レジスタ・フィールド：BFTST、BFCHG、BFCLR、BFSET＝000　上記以外＝デスティネーション・レジスタ
Doフィールド：0＝オフセットはイミディエイト値　1＝オフセットはデータ・レジスタにある。
Dwフィールド：0＝幅はイミディエイト値　1＝幅はデータ・レジスタにある。

PMOVE TT Registers

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
										実効アドレス					
1	1	1	1	0	0	0	0	0	0	モード			レジスタ		
0	0	0	Pレジスタ			R/W	FD	0	0	0	0	0	0	0	0

Pレジスタ・フィールド：
010－トランスペアレント変換レジスタ0
011－トランスペアレント変換レジスタ1
FDフィールド：0＝フラッシュがイネーブル　1＝フラッシュがディセーブル

PLOAD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
										実効アドレス					
1	1	1	1	0	0	0	0	0	0	モード			レジスタ		
0	0	1	0	0	0	R/W	0	0	0	0	FC				

FCフィールド：
10XXX－ファンクション・コードはビットXXXで指定される。
01DDD－ファンクション・コードはデータ・レジスタDDDにある。
00000－ファンクション・コードはSFCレジスタにある。
00001－ファンクション・コードはDFCレジスタにある。

PFLUSH

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
										実効アドレス					
1	1	1	1	0	0	0	0	0	0	モード			レジスタ		
0	0	1	モード			0	0	マスク			FC				

モード・フィールド：
001－すべてのエントリをフラッシュする。
100－ファンクション・コードによってのみフラッシュする。
110－ファンクション・コードと実効アドレスによってフラッシュする。
マスク・フィールド：ファンクション・コードのマスク。モード＝001の場合、マスクは000でなければならない。
FCフィールド：
10XXX－ファンクション・コードはビットXXXで指定される。
01DDD－ファンクション・コードはデータ・レジスタDDDにある。
00000－ファンクション・コードはSFCレジスタにある。
00001－ファンクション・コードはDFCレジスタにある。

PMOVE TC, SRP, and CRP Registers

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	0	0	実効アドレス モード			レジスタ		
0	1	0	Pレジスタ			R/$\overline{W}$	FD	0	0	0	0	0	0	0	0

Pレジスタ・フィールド：
- 000－TCレジスタ
- 010－SRPレジスタ
- 011－CRPレジスタ

FDフィールド：0＝フラッシュ・イネーブル　1＝フラッシュ・ディセーブル

PMOVE MMUSR Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	0	0	実効アドレス モード			レジスタ		
0	1	1	0	0	0	R/W	0	0	0	0	0	0	0	0	0

PTEST

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	0	0	実効アドレス モード			レジスタ		
1	0	0	レベル			R/W	A	レジスタ			FC				

レベル・フィールド：テーブルでのサーチの深さを指定

Aフィールド：
- 0－アドレス・レジスタにアドレスを返さない。
- 1－レジスタ・フィールドで指定されるアドレス・レジスタに最後にアクセスしたテーブルのアドレスを返す。

レジスタ・フィールド：最後のテーブルのアドレスを返すアドレス・レジスタ。
Aフィールド＝0のとき、このフィールドは000でなければならない。

FCフィールド：
- 10XXX－ファンクション・コードはビットXXXで指定される。
- 01DDD－ファンクション・コードはデータ・レジスタDDDにある。
- 00000－ファンクション・コードはSFCレジスタにある。
- 00001－ファンクション・コードはDFCレジスタにある。

コプロセッサ命令

cpGEN

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	CP-ID ≠000			0	0	0	実効アドレス モード			レジスタ		
コプロセッサ依存コマンド・ワード															
オプションの実効アドレスまたはコプロセッサ定義拡張ワード															

cpScc

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	CP-ID ≠000			0	0	1	実効アドレス モード			レジスタ		
0	0	0	0	0	0	0	0	0	0	コプロセッサ条件					
オプションの実効アドレスまたはコプロセッサ定義拡張ワード															

cpDBcc

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	CP-ID ≠000			0	0	1	0	0	1	レジスタ		
0	0	0	0	0	0	0	0	0	0	コプロセッサ条件					
オプションのコプロセッサ定義拡張ワード															
ディスプレースメント(16ビット)															

cpTRAPcc

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	CP-ID ≠000			0	0	1	1	1	1	モード		
0	0	0	0	0	0	0	0	0	0	コプロセッサ条件					
オプションのコプロセッサ定義拡張ワード															
オプションのワード															
またはロング・ワード・オペランド															

モード・フィールド：010＝ワード・オペランド　011＝ロング・ワード・オペランド
100＝ディスプレースメントなし

cpBcc

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	CP-ID ≠000			0	1	サイズ	コプロセッサ条件					
オプションのコプロセッサ定義拡張ワード															
ワードまたは															
ロング・ワード・ディスプレースメント															

サイズ・フィールド：0＝ワード・ディスプレースメント　1＝ロング・ワード・ディスプレースメント

cpSAVE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	CP-ID ≠000			1	0	0	実効アドレス モード			実効アドレス レジスタ		

cpRESTORE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	CP-ID ≠000			1	0	1	実効アドレス モード			実効アドレス レジスタ		

第3章 命令セット

第 4 章

処理状態

本章では、プロセッサの処理状態について解説します。ステータス・レジスタのスーパバイザ部分にあるビットの機能を述べ、例外状態に対してプロセッサがどのように処置を行なうかを説明します。

プロセッサは停止していないときには、通常または例外処理状態にあります。プロセッサが命令を実行しているとき、または命令あるいはオペランドをフェッチしているときには、通常の処理状態にあります。また、命令の結果を格納したりコプロセッサとやりとりを行なっているときにも通常の処理状態にあります。

注：例外処理とは、プログラムの通常処理状態から、システム・ルーチン、割込みルーチン、そしてその他の割込みハンドラへの移行中の状態を指します。例外処理には、すべてのスタッキング操作、例外ベクタのフェッチ、および例外による命令パイプの充てんが含まれます。例外処理は、例外ハンドラ・ルーチンの最初の命令の実行が開始されると完了します。

プロセッサは、割込みの認識応答、命令のトレース、またはトラップの結果により、あるいはその他の例外的な状態の発生により、例外処理状態に入ります。例外は特定の命令を実行することによって、あるいは命令の実行中に生じる異常状態によって発生します。また、割込み、バス・エラー、およびいくつかのコプロセッサ応答によっても発生します。例外処理では、例外を処理するハンドラやルーチンに効率よく制御を移せるようになっています。

例外処理状態にあるときに、プロセッサがバス・エラーを受け取ったり、アドレス・エラーを発生すると、致命的なシステム障害が発生します。このような障害が発生するとプロセッサは停止してしまいます。たとえば、あるバス・エラーの例外処理を実行している間に別のバス・エラーが発生したが、MC68030がまだ通常処理に移行しておらず、マシンの内部状態のセーブを完了していない場合、プロセッサはシステムが使用不能になったものと判断してホルトします。ホルト状態のプロセッサを再スタートさせるには、外部リセットしか方法はありません(プロセッサがSTOP命令を実行して入るストップ状態は通常の処理状態の特殊ケースで、バス・サイクルは発生しません。これはストップ状態であり、ホルト状態とは異なります)。

4.1　特権レベル

プロセッサは、ユーザ・レベルとスーパバイザ・レベルの2つの特権レベルのいずれかで動作します。スーパバイザ・レベルのほうがユーザ・レベルより高い特権レベルです。特権レベルの低いユーザ・レベルでは、実行できない命令がありますが、スーパバイザ・レベルではすべての命令を実行できます。これによって、スーパバイザとユーザを区別することができるため、スーパバイザは権限のないアクセスからシステム資源を保護することができます。プロセッサはステータス・レベル

のSビットで指示される特権レベルを使用して、ユーザまたはスーパバイザ特権レベルを選択し、スタック操作にユーザ・スタック・ポインタまたはスーパバイザ・スタック・ポインタを選択することができます。プロセッサは、ファンクション・コードでバス・アクセス(スーパバイザまたはユーザ・モード)を識別するため、スーパバイザとユーザ・モードの違いを認識できます。メモリ管理ユニットは特権レベルの表示を用いて、メモリ・アクセスの制御と変換を行ない、ユーザ・プログラムのアクセスからスーパバイザ・コード、データ、および資源を保護します。

多くのシステムでは、プログラムの大部分がユーザ・レベルで実行されます。ユーザ・プログラムは、それ自身のコードとデータ領域にしかアクセスできず、他の情報へのアクセスが制限されることもあります。OSは通常スーパバイザ特権レベルで実行されます。OSはすべての資源にアクセスでき、ユーザ・レベルのプログラムのためのオーバヘッド・タスクを実行し、それらの活動を調整します。

4.1.1　スーパバイザ特権レベル

スーパバイザ・レベルは上位の特権レベルです。この特権レベルは、ステータス・レジスタのSビットによって設定されます。Sビットがセットされている場合、プロセッサはスーパバイザ特権レベルにあり、すべての命令を実行できます。スーパバイザ・レベルで実行されている命令に対するバス・サイクルは、通常“スーパバイザ参照”に分類され、FC0-FC2のファンクション・コード値でスーパバイザ・アドレス空間を識別します。

マルチタスキングのOSでは、各ユーザ・タスクに関連付けられたスーパバイザ・スタック空間をもち、別に割込み関連タスク用のスタック空間を用意するほうがより効率的です。MC68030はマスタおよび割込みの2つのスーパバイザ・スタックを備えており、ステータス・レジスタのMビットでこの2つのうちどちらをアクティブにするかを選択します。Mビットが1にセットされているとき、スーパバイザ・スタック・ポインタ参照(暗黙に、あるいはアドレス・レジスタA7を指定したもの)は、マスタ・スタック・ポインタMSPにアクセスします。OSは各タスクのMSPがスーパバイザ・データ空間のタスク関連領域を指すように設定します。これによって、タスク関連のスーパバイザ動作を、現在実行中のタスクと偶然に一致する可能性のある非同期のI/O関連スーパバイザ・タスクと分離します。マスタ・スタック(MSP)は、現在実行中の各ユーザ・タスクに関するタスク制御情報を個別に維持することができ、タスク・スイッチが行なわれたときにソフトウェアがMSPポインタを更新することにより、タスク関連のスタック情報を効率よく転送するための手段を提供します。他のスーパバイザ・スタック(ISP)は、割込み処理ルーチンの要求に応じて、割込み制御情報および作業用空間として使用することができます。

Mビットがクリアされているとき、MC68030はスーパバイザ特権レベルの割込みモードにあり、MC68000、MC68008、およびMC68010のスーパバイザ・モードと同じ動作を実行します(プロセッサはリセット動作後にこのモードになります)。このモードでは、スーパバイザ・スタック・ポインタ参照はすべて割込みスタック・ポインタ(ISP)を使用します。

ステータス・レジスタのMビットの値は、特権命令の実行には影響を与えません。マスタおよび割込みモードの両方がスーパバイザ特権レベルにあります。Mビットに影響を与える命令は、MOVE to SR、ANDI to SR、EORI to SR、ORI to SR およびRTEです。また、プロセッサは割込みに対する例外処理の過程で、Mビットの状態をセーブしたのち、SRのMビットをクリアします。

例外処理はすべてスーパバイザ特権レベルで実行されます。例外処理中に発生するすべてのバス・サイクルが“スーパバイザ参照”であり、スタック・アクセスはどれもアクティブ・スーパバイザ・スタック・ポインタを使用します。

4. 1. 2　ユーザ特権レベル

ユーザ・レベルは下位の特権レベルです。この特権レベルはステータス・レジスタのSビットによって判定されます。Sビットがクリアされている場合、プロセッサはユーザ・レベルで命令を実行します。

ほとんどの命令は、ユーザ・レベルとスーパバイザ・レベルのいずれかの特権レベルで実行されますが、システムに対して重要な影響を与える一部の命令は特権化されていて、スーパバイザ・レベルでしか実行できません。たとえば、ユーザ・プログラムでは、STOP命令やRESET命令は実行できません。ユーザ・プログラムが管理された方法以外でスーパバイザ特権レベルに入るのを防止するために、ステータス・レジスタのSビットを変更できる命令は特権化されています。ユーザ・プログラムでTRAP #n命令を使用すれば、管理された方法によってOSのサービスにアクセスすることができます。

ユーザ・レベルで実行される命令のバス・サイクルは“ユーザ参照”に分類され、FC0～FC2のファンクション・コード値でユーザ・アドレス空間を指定します。プロセッサのメモリ管理ユニットは、イネーブルになっているときには、ファンクション・コード値を使用して、ユーザとスーパバイザの動作を区別し、アドレス空間で保護された部分へのアクセスを管理することができます。プロセッサがユーザ・レベルにあるときには、暗黙にシステム・スタック・ポインタを参照したり、明示的にアドレス・レジスタ7(A7)を参照すると、すべてユーザ・スタック・ポインタ(USP)を参照することになります。

4. 1. 3　特権レベルの変更

ユーザ・レベルからスーパバイザ・レベルに変更するには、プロセッサに例外処理を行なわせる条件の1つが発生しなければなりません。例外状態を発生させることによって、ユーザ・レベルからスーパバイザ・レベルに切り換え、さらにマスタ・モードから割込みモードに切り換えることができます。例外処理は、ステータス・レジスタのSビットとMビットの現在値を(ステータス・レジスタの残りの部分とともに)、アクティブ・スーパバイザ・スタックにセーブし、Sビットをセットしてプロセッサを強制的にスーパバイザ・レベルにします。処理中の例外が割込みであり、かつMビットがセットされている場合はMビットがクリアされ、プロセッサは割込みモードになります。命令の実行はスーパバイザ・レベルで進行し、例外状態の処理が行なわれます。

スーパバイザ・レベルからユーザ・レベルに戻るには、システム・ルーチンで、MOVE to SR、ANDI to SR、EORI to SR、ORI to SR、またはRTEを実行しなければなりません。MOVE、ANDI、EORI、およびORI to SR、そしてRTE命令は、スーパバイザ・レベルで実行でき、ステータス・レジスタのSビットの変更が可能です。これらの命令を実行した後、命令パイプラインはフラッシュされ、所定のアドレス空間から再充てんされます。この動作は$\overline{\text{REFILL}}$信号のアサートにより外部に通知されます。

RTE命令は例外が発生したときに実行されていたプログラムに復帰します。RTE命令は、スーパバイザ・スタックにセーブされている例外スタック・フレームを復元します。スタックの先頭にあるフレームが割込み、トラップ、または命令例外によって生成されたものである場合、RTE命令はステータス・レジスタとプログラム・カウンタの内容をスーパバイザ・スタックにセーブされている値に復元します。ついで、プロセッサは復元されたプログラム・カウンタ・アドレスから、ステータス・レジスタのSビットの値によって決まる特権レベルで実行を継続します。スタックの先頭にあるフレームが、バス・フォールト(バス・エラーまたはアドレス・エラー例外)によって生成されたものである場合、RTE 命令はセーブされているプロセッサのすべての内部状態をスタックから復元します。

表4-1 アドレス空間のエンコーディング

FC2	FC1	FC0	アドレス空間
0	0	0	(未定義、予約) *
0	0	1	ユーザ・データ空間
0	1	0	ユーザ・プログラム空間
0	1	1	(未定義、予約) *
1	0	0	(未定義、予約) *
1	0	1	スーパバイザ・データ空間
1	1	0	スーパバイザ・プログラム空間
1	1	1	CPU空間

*アドレス空間3はユーザ定義用に予約されており、0および4は将来使用のためモトローラにより予約されています。

4. 2　アドレス空間の種類

プロセッサは要求されるアクセスの種類に応じて、ファンクション・コード信号により各バス・サイクルに対応するターゲット・アドレス空間を指定します。スーパバイザ/ユーザおよびプログラム/データを区別するだけでなく、割込みアクノリッジ・サイクルなどの特殊プロセッサ・サイクルを識別することもできます。また、メモリ管理ユニットは、アクセスを制御し適切なアドレス変換を実行することができます。表4-1に、MC68030で定義されているアクセスの種類と対応するファンクション・コードFC0-FC2の値をリストします。

ユーザ・プログラムおよびデータ・アクセス用のメモリ・ロケーションは、あらかじめ定義されていません。また、スーパバイザ・データ空間のロケーションも定義されていません。リセット時には、スーパバイザ・プログラム空間のメモリ・ロケーション0から始まる最初の2つのロング・ワードが、プロセッサの初期化に使用されます。それ以外、MC68030で明示的に定義されているメモリ・ロケーションはありません。

ファンクション・コード$7([FC2:FC0] =111)は、CPUアドレス空間を選択します。これは特殊なアドレス空間で、命令やオペランドは含まれていませんが、特殊プロセッサ機能のために確保されています。プロセッサはこの空間にアクセスすることにより、特殊な目的をもつ外部デバイスと交信を行ないます。たとえば、MC68000ファミリのすべてのプロセッサは、CPU 空間を割込みアクノリッジ・サイクルに使用します。また、MC68020とMC68030は、ブレークポイント・アクノリッジとコプロセッサ操作にも、CPU 空間アクセスを行ないます。

スーパバイザ・プログラムはMOVES命令を使用して、ユーザ空間とCPUアドレス空間を含む全アドレス空間にアクセスすることができます。MOVES命令を使用して、CPU空間バス・サイクルを発生させることができますが、これによってシステムの正常な動作に妨害を与えることがありますので、MOVESを使用してCPU空間にアクセスする場合には注意が必要です。

4. 3　例外処理

例外は正常な処理の流れを変える特殊状態と定義されます。内部と外部の両方の状態によって例外が発生します。例外を発生する外部状態は、外部デバイスからの割込み、バス・エラー、コプロセッサ検出エラー、およびリセットがあります。命令、アドレス・エラー、トレース、およびブレークポイントなどが、例外を発生する内部状態です。TRAP、TRAPcc、TRAPV、cpTRAPcc、CHK、CHK2、RTEおよびDIV命令はすべて、通常の実行の一環として例外を生成することができます。そ

の他、不当命令、特権違反、およびコプロセッサ・プロトコル違反も例外を発生します。

例外処理は、プログラムの通常処理から例外状態に要求される処理への移行であり、例外ベクタ・テーブルと例外スタック・フレームが関係します。本章では、ベクタ・テーブルと一般的な例外スタック・フレームについて説明します。例外処理の詳細については、「第8章　例外処理」を参照してください。また、コプロセッサが検出する例外の詳細は、「第10章　コプロセッサ・インタフェースの説明」で説明しています。

4.3.1　例外ベクタ

ベクタ・ベース・レジスタは、1024バイトの例外ベクタ・テーブルのベース・アドレスを保持しており、この中には256の例外ベクタが含まれています。例外ベクタには、例外処理が完了すると実行を開始するルーチンのメモリ・アドレスが入っています。これらのルーチンは、対応する例外に応じた一連の操作を実行します。例外ベクタにはメモリ・アドレスが入っているため、リセット・ベクタを除いて長さはすべて1ロング・ワードです。リセット・ベクタは、割込みスタック・ポインタの初期化に使用するアドレス、およびプログラム・カウンタの初期化に使用するアドレスの2つで構成されています。

例外ベクタのアドレスは、8ビットのベクタ番号とベクタ・ベース・レジスタ(VBR)で作られます。例外ベクタ番号には外部デバイスから得られるものとプロセッサが自動的に供給するものがあります。プロセッサは、このベクタ番号を4倍にしてベクタ・オフセットを計算し、それをVBRに加算します。その合計がベクタのメモリ・アドレスになります。リセット・ベクタ以外の例外ベクタはすべて、スーパバイザ・データ空間にあります。リセット・ベクタは、スーパバイザ・プログラム空間にあります。プロセッサのメモリ・マップでは、初期のリセット・ベクタだけが固定されています。初期化が完了してしまうと、固定された割当てはなくなります。VBRはベクタ・テーブルのベース・アドレスを与えますので、ベクタ・テーブルはメモリのどこにでも配置することができます。OSで実行される個々のタスクに対してダイナミックに再配置することも可能です。例外処理の詳細は、「第8章　例外処理」で説明しています。また、表8-1に例外ベクタの割当てを示します。

4.3.2　例外スタック・フレーム

例外処理では、プロセッサの現在のコンテキストのうち書き換えられる可能性のある部分のほとんどがセーブされます。このコンテキストは、例外スタック・フレームとよぶフォーマットで構成されています。この情報には、常にステータス・レジスタのコピー、プログラム・カウンタ、ベクタのベクタ・オフセット、およびフレーム・フォーマット・フィールドが含まれています。フレーム・フォーマット・フィールドは、スタック・フレームのタイプを識別します。RTE命令は、フォーマット・フィールドの値を使用して、スタック・フレームに格納されている情報を適切に復元し、スタック空間の割当て解除を行ないます。例外スタック・フレームの一般的な形式を図4-1に示します。例外スタック・フレームの完全なリストが「第8章　例外処理」にありますので参照してください。

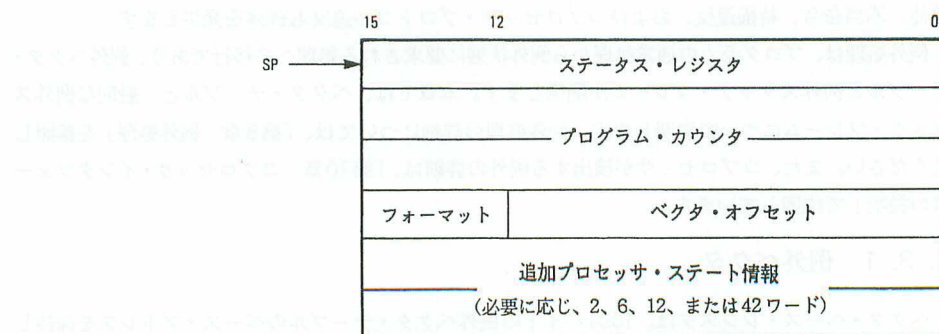



図4-1　一般的な例外スタック・フレーム

第 5 章

信号の説明

本章では、入力信号と出力信号を図5-1 に示す機能グループごとに簡単に説明します。各信号は短いパラグラフで説明してあり、その機能についての詳細が記載されている章がある場合は、それを参照するようになっています。

注：以下、信号を特定の状態に置くことを表わすのに、アサーションまたはアサート、およびネゲーションまたはネゲートという用語を頻繁に使用しています。アサーションとアサートはアクティブまたは真の信号を意味します。ネゲーションとネゲートは非アクティブまたは偽の信号を示します。これらの用語は、信号の電圧レベル("H" または "L")とは関係なく使用されます。

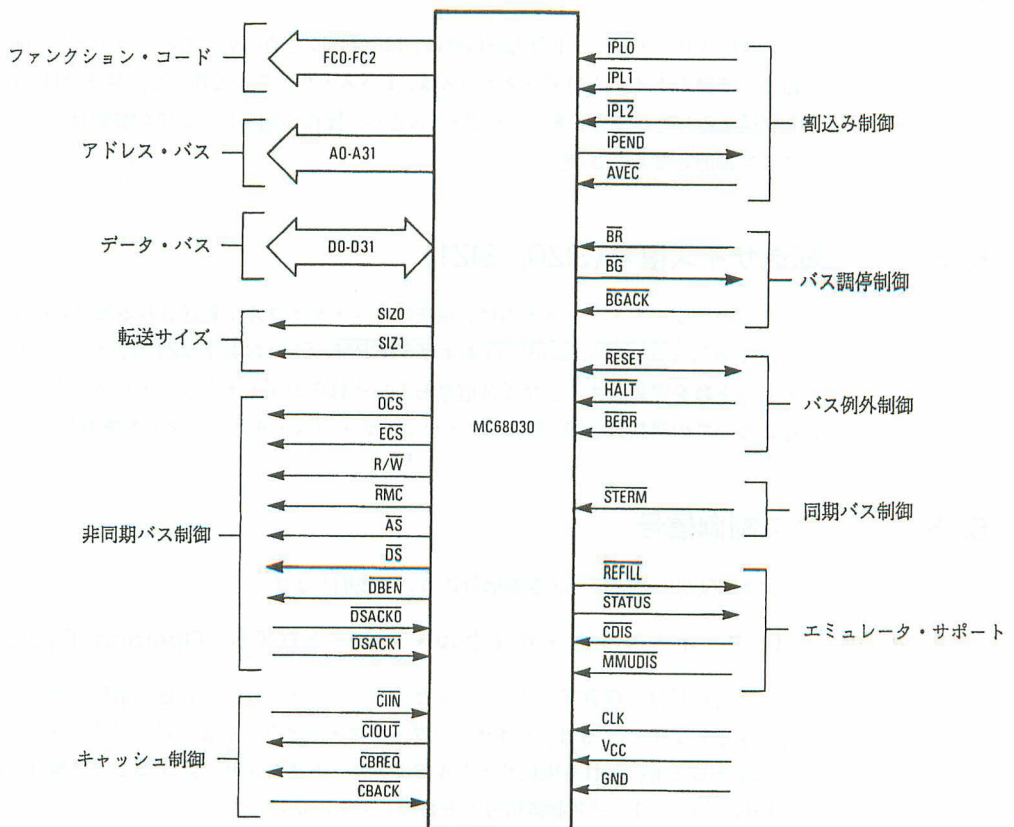



図5-1　信号の機能グループ

5. 1　信号名

MC68030の入力および出力信号を表5-1に示します。各信号の信号名とニーモニックが簡単な説明とともに記載してあります。各信号の詳細については、本文の該当するパラグラフとそこに示されている関連機能の説明の項を参照してください。

表5-1に示す信号の保証タイミング仕様は、「13章　電気的特性」に記載されています。

5. 2　ファンクション・コード信号(FC0～FC2)

これらのスリー・ステート出力は、現在のバス・サイクルのアドレス空間を識別します。表4－1にファンクション・コード信号と特権レベルおよびアドレス空間の関係を示しました。詳細については、「4.2　アドレス空間の種類」を参照してください。

5. 3　アドレス・バス(A0～A31)

これらのスリー・ステート出力は、CPUアドレス空間を除く現在のバス・サイクルのアドレスを示します。CPUアドレス空間の詳細は、「4. 2　アドレス空間の種類」を参照してください。A31が最上位アドレス信号です。アドレス・バスとバス操作の関係については、「7. 1. 2　アドレス・バス」を参照してください。

5. 4　データ・バス(D0～D31)

これらのスリー・ステート双方向信号は、MC68030と他のすべてのデバイスとの間の汎用データ経路として働きます。このデータ・バスは、1バス・サイクルで8、16、24または32ビットのデータを転送することができます。データ・バスとバス操作の関係についての詳細は、「7. 1. 4　データ・バス」を参照してください。

5. 5　転送サイズ信号(SIZ0、SIZ1)

これらのスリー・ステート出力は、現在のバス・サイクルで転送される残りのバイト数を示します。A0、A1、$\overline{\text{DSACK0}}$、$\overline{\text{DSACK1}}$および$\overline{\text{STERM}}$、SIZ0およびSIZ1で、データ・バス上で転送されるビット数を定義します。サイズ信号およびそれらのダイナミック・バス・サイジングにおける用途についての詳細は、「7. 2. 1　ダイナミック・バス・サイジング」を参照してください。

5. 6　バス制御信号

次にMC68030の同期バス制御信号について説明します。

5. 6. 1　オペランド・サイクル・スタート($\overline{\text{OCS}}$: Operand Cycle Start)

この出力信号は、命令プリフェッチまたはデータ・オペランド転送の最初の外部バス・サイクルの開始を示します。$\overline{\text{OCS}}$は、ダイナミック・バス・サイジングまたはオペランドのミスアラインメントに起因して実行される後続サイクルではアサートされません。$\overline{\text{OCS}}$とバス操作の関係についての詳細は、「7. 1. 1　バス制御信号」を参照してください。

表5-1　信号一覧

信　号　名	ニーモニック	機　　　能
ファンクション・コード	FC0-FC2	各バス・サイクルのアドレス空間を識別するために使用する3ビットのファンクション・コード
アドレス・バス	A0-A31	32ビットのアドレス・バス
データ・バス	D0-D31	1バス・サイクル当たり8、16、24、または32ビットを転送するのに使用する32ビットのデータ・バス
サイズ	SIZ0/SIZ1	このサイクルで転送する残りのバイト数を示す。これらの信号は、A0およびA1とともにデータ・バスのアクティブなセクションを決める。
オペランド サイクル・スタート	$\overline{OCS}$	$\overline{OCS}$はオペランド転送の第1バス・サイクルでのみアサートされる点を除いて、$\overline{ECS}$と同じ機能をもつ。
外部サイクル・スタート	$\overline{ECS}$	バス・サイクルの開始を示す。
リード/ライト	R/$\overline{W}$	プロセッサがリードまたはライトを行なうときにバス転送の方向を定義する。
リード・モディファイ・ライト・サイクル	$\overline{RMC}$	現行バス・サイクルが不可分のリード・モディファイ・サイクルの一部であることを示す。
アドレス・ストローブ	$\overline{AS}$	バス上に有効なアドレスがあることを示す。
データ・ストローブ	$\overline{DS}$	外部デバイスがデータ・バスに有効なデータを出力すべきこと、あるいはMC68030がデータ・バスに有効なデータを出力していることを示す。
データ・バッファ・イネーブル	$\overline{DBEN}$	外部データ・バッファにイネーブル信号を供給する。
データ転送および サイズ・アクノリッジ	$\overline{DSACK0}$/ $\overline{DSACK1}$	要求されたデータ転送操作が完了したことを示すバス応答信号。また、この2本のラインはサイクルはサイクルごとに外部バス・ポートのサイズを示し、非同期転送に使用される。
同期ターミネーション	$\overline{STERM}$	ポート・サイズが32ビットであり、クロックの次の立下りエッジでデータのラッチが可能なことを示すバス応答信号。
キャッシュ インヒビット・イン	$\overline{CIIN}$	MC68030の命令キャッシュおよびデータ・キャッシュにデータをロードしないようにするための信号。
キャッシュ インヒビット・アウト	$\overline{CIOUT}$	ATCエントリまたはTTxレジスタのCIビットを反映する。外部キャッシュにこれらのアクセスを無視するよう指示する。
キャッシュ・バースト要求	$\overline{CBREQ}$	命令キャッシュまたはデータ・キャッシュに対するバースト要求を示す。
キャッシュ・バースト・アクノリッジ	$\overline{CBACK}$	アクセスされたデバイスがバースト・モードで動作可能なことを示す。
割込み優先レベル	$\overline{IPL0}$-$\overline{IPL2}$	プロセッサにエンコードされた割込みレベルを与える信号。
割込みペンディング	$\overline{IPEND}$	割込みが保留されていることを示す。
オートベクタ	$\overline{AVEC}$	割込みアクノリッジ・サイクル中にオートベクタを要求するための信号。
バス要求	$\overline{BR}$	外部デバイスがバス・マスタになることを要求していることを示す信号。
バス許可	$\overline{BG}$	外部デバイスがバス・マスタになれることを示す信号。
バス許可アクノリッジ	$\overline{BGACK}$	外部デバイスがバス・マスタになったことを示す。
リセット	$\overline{RESET}$	システム・リセット
ホルト	$\overline{HALT}$	プロセッサにバス操作を停止させるための信号。
バス・エラー	$\overline{BERR}$	誤ったバス操作を行なおうとしていることを示す。
キャッシュ・ディセーブル	$\overline{CDIS}$	エミュレータをサポートするために、オンチップ・キャッシュをダイナミックにディセーブルするための信号。
MMUディセーブル	$\overline{MMUDIS}$	MMUの変換メカニズムをダイナミックにディセーブルするための信号。
パイプの再充てん	$\overline{REFILL}$	MC68030がパイプの充てんを開始したことを示す。
マイクロシーケンサ・ステータス	$\overline{STATUS}$	マイクロシーケンサの状態を示す。
クロック	CLK	プロセッサへのクロック入力
電源	V_{CC}	電源
グランド	GND	グランド接続

第5章　信号の説明

5. 6. 2　外部サイクル・スタート($\overline{\text{ECS}}$: External Cycle Start)

この出力信号はバス・サイクルの開始を示します。$\overline{\text{ECS}}$とバス操作の関係については、「7. 1. 1 バス制御信号」を参照してください。

5. 6. 3　リード/ライト(R/$\overline{\text{W}}$: Read/Write)

このスリー・ステート出力信号はバス・サイクルの種類を定義します。“H”レベルはリード・サイクルを示し、“L”レベルはライト・サイクルを示します。R/$\overline{\text{W}}$とバス操作の関係については、「7. 1. 1 バス制御信号」を参照してください。

5. 6. 4　リード・モディファイ・ライト・サイクル($\overline{\text{RMC}}$: Read-Modify-Write Cycle)

このスリー・ステート出力信号は、現在のバス・サイクルが不可分のリード・モディファイ・ライト・サイクルであることを示します。この信号はリード・モディファイ・ライト操作中はアサートされたままです。$\overline{\text{RMC}}$とバス操作の関係については、「7. 1. 1　バス制御信号」を参照してください。

5. 6. 5　アドレス・ストローブ($\overline{\text{AS}}$: Address Strobe)

このスリー・ステート出力信号は、アドレス・バス上に有効なアドレスがあることを示します。$\overline{\text{AS}}$がアサートされているときには、ファンクション・コード、サイズ、およびリード/ライト信号も有効です。$\overline{\text{AS}}$とバス操作の関係については、「7. 1. 3　アドレス・ストローブ」を参照してください。

5. 6. 6　データ・ストローブ($\overline{\text{DS}}$: Data Strobe)

このスリー・ステート出力信号は、リード・サイクルでは外部デバイスがデータ・バスに有効なデータを出力しなければならないことを示します。ライト・サイクルでは、MC68030が有効なデータをデータ・バスに出力していることを示します。2クロック同期ライト・サイクルでは、MC68030は$\overline{\text{DS}}$をアサートしません。$\overline{\text{DS}}$とバス操作の関係については、「7. 1. 5　データ・ストローブ」を参照してください。

5. 6. 7　データ・バッファ・イネーブル($\overline{\text{DBEN}}$: Data Buffer Enable)

この出力は外部データ・バッファのイネーブル信号です。この信号はすべてのシステムに必要なものではありません。この信号のタイミングは、2クロック同期バス・サイクルをサポートするシステムでは使用できないことがあります。$\overline{\text{DBEN}}$とバス操作の関係についての詳細は、「7. 1. 6　データ・バッファ・イネーブル」を参照してください。

5. 6. 8　データ転送およびサイズ・アクノリッジ($\overline{\text{DSACK0}}$、$\overline{\text{DSACK1}}$: Data Transfer and Size Acknowledge)

これらの入力は要求されたデータ転送が完了したことを示します。また、各サイクルの終了時点で、外部バス・ポートのサイズを示します。これらの信号は非同期バス・サイクルにだけ有効です。これらの信号とダイナミック・バス・サイジングの関係については、「7. 1. 7　バス・サイクル・ターミネーション信号」を参照してください。

5. 6. 9　同期ターミネーション($\overline{\text{STERM}}$: Synchronous Termination)

この入力はバス・ハンドシェイク信号で、アドレス指定されたポート・サイズが32ビットであり、

リード・サイクルでは次の立下りクロック・エッジでデータをラッチすることを示します。この信号は同期操作にだけ有効です。$\overline{\text{STERM}}$とバス操作の関係についての詳細は、「7. 1. 7　バス・サイクル・ターミネーション信号」を参照してください。

5. 7　キャッシュ制御信号

次にオンチップ・キャッシュに関連する信号について説明します。

5. 7. 1　キャッシュ・インヒビット・インプット($\overline{\text{CIIN}}$: Cache Inhibit Input)

この入力信号は、MC68030の命令キャッシュおよびデータ・キャッシュにデータがロードされないようにします。これは同期入力信号であり、バス・サイクルごとに解釈されます。$\overline{\text{CIIN}}$はライト・サイクル中は無視されます。$\overline{\text{CIIN}}$とオンチップ・キャッシュの関係については、「6. 1　オンチップ・キャッシュの構成と操作」を参照してください。

5. 7. 2　キャッシュ・インヒビット・アウトプット($\overline{\text{CIOUT}}$: Cache Inhibit Output)

このスリー・ステート出力信号は、参照論理アドレスに対するアドレス変換キャッシュ・エントリのCIビットの状態を反映するもので、外部キャッシュがバス転送を無視しなければならないことを示します。参照論理アドレスが透過変換で指定される領域内の場合は、該当する透過変換レジスタのCIビットで$\overline{\text{CIOUT}}$の状態を制御します。アドレス変換キャッシュと透過変換についての詳細は、「第9章　メモリ管理ユニット」を参照してください。また、内部キャッシュに対する$\overline{\text{CIOUT}}$の影響については、「第6章　オンチップ・キャッシュ・メモリ」を参照してください。

5. 7. 3　キャッシュ・バースト・リクエスト($\overline{\text{CBREQ}}$: Cache Burst Request)

このスリー・ステート出力信号は、命令キャッシュまたはデータ・キャッシュの1ラインを充てんするためのバースト・モード操作を要求します。情報の充てんについては、「6. 1. 3　キャッシュの充てん」、そしてバースト・モード操作に関するバス・サイクルの情報については、「7. 3. 7　バースト操作サイクル」を参照してください。

5. 7. 4　キャッシュ・バースト・アクノリッジ($\overline{\text{CBACK}}$: Cache Burst Acknowledge)

この入力信号は、アクセスされたデバイスがバースト・モードで動作可能であり、命令キャッシュまたはデータ・キャッシュに、最低もう1つのロング・ワードを供給できることを示します。バースト・モード操作については、「7. 3. 7　バースト操作サイクル」を参照してください。

5. 8　割込み制御信号

次にMC68030の割込み制御信号について説明します。

5. 8. 1　割込み優先レベル信号

これらの入力信号は、割込み状態の表示と周辺デバイスまたは優先順位回路からの割込みレベルのエンコーディングを与えます。$\overline{\text{IPL2}}$がレベル番号の最上位ビットです。

たとえば、$\overline{\text{IPLn}}$信号は負論理であるため$\overline{\text{IPL0}}$-$\overline{\text{IPL2}}$が＄5を示すときは、割込みレベル2の割込み要求に対応します。MC68030の割込みに関する情報は、「8. 1. 9　割込み例外」を参照してくだ

さい。

5. 8. 2　割込みペンディング($\overline{\text{IPEND}}$: Interrupt　Pending)

この出力信号は割込み要求が内部で認識され、ステータス・レジスタ(SR)の現在の割込み優先マスクより優先順位が高いことを示します。この出力は、次の命令境界についてのプロセッサの操作を予測するために、外部デバイス(たとえば、コプロセッサや他のバス・マスタ)が使用するためのものです。割込みについては、「8. 1. 9　割込み例外」を参照してください。また、割込みに関連するバス情報については、「7. 4. 1　割込みアクノリッジ・バス・サイクル」を参照してください。

5. 8. 3　オートベクタ($\overline{\text{AVEC}}$: Autovector)

この入力信号は、MC68030が割込みアクノリッジ・サイクル中にオートベクタを生成しなければならないことを示します。オートベクタについての詳細は、「7. 4. 1. 2　オートベクタ割込みアクノリッジ・サイクル」を参照してください。

5. 9　　バス調停制御信号

次にシステム内のどのデバイスがバス・マスタになるかを決定するのに使用する、3つのバス調停制御信号について説明します。

5. 9. 1　バス要求($\overline{\text{BR}}$: Bus　Request)

この入力信号は、外部デバイスがバス・マスタになる必要があることを示します。この入力は通常“ワイヤードOR”されています(ただし、必ずしもオープン・コレクタ・デバイスで構成する必要はありません)。詳細については、「7. 7　バス調停」を参照してください。

5. 9. 2　バス許可($\overline{\text{BG}}$: Bus　Grant)

この出力信号は、MC68030が現在のプロセッサ・バス・サイクルの完了時にバスの制御権を放棄することを示します。詳細については、「7. 7. 2　バス許可」を参照してください。

5. 9. 3 バス許可アクノリッジ($\overline{\text{BGACK}}$: Bus　Grant　Acknowledge)

この入力信号は、外部デバイスがバス・マスタになったことを示します。詳細については、「7. 7. 3　バス許可アクノリッジ」を参照してください。

5. 10　　バス例外制御信号

次にMC68030のバス例外制御信号について説明します。

5. 10. 1　リセット($\overline{\text{RESET}}$)

この双方向オープン・ドレイン信号は、システム・リセットを開始するのに使用します。外部リセット信号は、MC68030およびすべての外部デバイスをリセットします。

プロセッサからのリセット信号(RESET命令の動作の一部としてアサートされる)は、外部デバイスだけをリセットします。プロセッサの内部状態は変更されません。リセット・バス操作の説明は、「7. 8　リセット動作」を参照してください。また、リセット例外に関する情報は、「8. 1. 1 リセット例外」を参照してください。

5. 10. 2　ホルト($\overline{\mathrm{HALT}}$)

ホルト信号はプロセッサがバス動作を中断しなければならないことを示します。また、$\overline{\mathrm{BERR}}$とともに使用したときは、プロセッサが現在のバス・サイクルを再試行しなければならないことを示します。バス操作への$\overline{\mathrm{HALT}}$の影響については、「7. 5 バス例外制御サイクル」を参照してください。

5. 10. 3　バス・エラー($\overline{\mathrm{BERR}}$：Bus　Error)

バス・エラー信号は無効なバス操作を行なっていることを示します。また、$\overline{\mathrm{HALT}}$とともに使用したときは、プロセッサが現在のサイクルを再試行しなければならないことを示します。バス操作への$\overline{\mathrm{BERR}}$の影響については、「7. 5　バス例外制御サイクル」を参照してください。

5. 11　エミュレータ・サポート信号

次の信号は、エミュレータにオンチップ・キャッシュとメモリ管理ユニットをディセーブルする手段を提供し、また内部ステータス情報を与えることによって、エミュレーションのサポートを行ないます。エミュレーションのサポートについての詳細は、「第12章　アプリケーション情報」を参照してください。

5. 11. 1　キャッシュ・ディセーブル(CDIS：Cache　Disable)

キャッシュ・ディセーブル信号は、オンチップ・キャッシュをダイナミックにディセーブルして、エミュレータ・サポートを援助します。キャッシュに関しては、「6. 1　オンチップ・キャッシュの構成と操作」を参照してください。エミュレータでこの信号を使用する方法については、「第12章　アプリケーション情報」を参照してください。$\overline{\mathrm{CDIS}}$はデータ・キャッシュおよび命令キャッシュのフラッシュは行ないません。エントリはもとのままで、$\overline{\mathrm{CDIS}}$がネゲートされると、再び使用できるようになります。

5. 11. 2　MMUディセーブル($\overline{\mathrm{MMUDIS}}$：MMU　Disable)

MMUディセーブル信号は、MMUによるアドレス変換をダイナミックにディセーブルします。アドレス変換については、「9. 4　アドレス変換キャッシュ」を参照してください。この信号をエミュレータで使用する方法については、「第12章　アプリケーション情報」を参照してください。$\overline{\mathrm{MMUDIS}}$をアサートしても、アドレス変換キャッシュ(ATC)はフラッシュされません。$\overline{\mathrm{MMUDIS}}$がネゲートされると、再びATCエントリが使用できるようになります。

5. 11. 3　パイプラインの再充てん($\overline{\mathrm{REFILL}}$：Pipeline　Refill)

パイプラインの再充てん信号は、MC68030が内蔵する命令パイプラインの再充てんを開始したことを示します。この信号をエミュレータで使用する方法については、「第12章　アプリケーション情報」を参照してください。

5. 11. 4　内部マイクロシーケンサ・ステータス($\overline{\mathrm{STATUS}}$：Internal Microsequencer　Status)

マイクロシーケンサ・ステータス信号は、内部マイクロシーケンサの状態を示します。この信号がアサートされるクロック数によって、命令の境界、保留中の例外、およびホルト状態を示します。この信号をエミュレータで使用する方法については、「第12章　アプリケーション情報」を参照してください。

第5章　信号の説明

5. 12　クロック(CLK)

クロック信号はMC68030へのクロック入力です。この信号はTTLコンパチブルです。クロック生成に関する要点については、「第12章　アプリケーション情報」、そしてクロックの電気的仕様については、「第13章　電気的特性」を参照してください。

5. 13　電源の接続

MC68030はグランドに対し正電位であるV_{CC}電源に接続しなければなりません。V_{CC}接続はプロセッサ各部に十分な電流を供給するために、グループごとに配置されています。また、グランド接続も同様にグループごとに配置されています。「第14章　注文情報および機械的データ」にV_{CC}およびグランド接続のグループを記載しています。また、「第12章　アプリケーション情報」に代表的な電源インタフェースを示します。

5. 14　信号の要約

これまで説明した信号の電気的特性を表5-2にまとめて示します。

表5-2　信号一覧

信号機能	信号名	入力/出力	ｱｸﾃｨﾌﾞ･ｽﾃｰﾄ	スリー・ステート
ファンクション・コード	FC0-FC2	出力	“H”	Yes
アドレス・バス	A0-A31	出力	“H”	Yes
データ・バス	D0-D31	入出力	“H”	Yes
転送サイズ	SIZ0/SIZ1	出力	“H”	Yes
オペランド・サイクル	$\overline{\mathrm{OCS}}$	出力	“L”	No
外部サイクル・スタート	$\overline{\mathrm{ECS}}$	出力	“L”	No
リード/ライト	R/$\overline{\mathrm{W}}$	出力	“H” / “L”	Yes
リード・モディファイ・ライト・サイクル	$\overline{\mathrm{RMC}}$	出力	“L”	Yes
アドレス・ストローブ	$\overline{\mathrm{AS}}$	出力	“L”	Yes
データ・ストローブ	$\overline{\mathrm{DS}}$	出力	“L”	Yes
データ・バッファ・イネーブル	$\overline{\mathrm{DBEN}}$	出力	“L”	Yes
データ転送およびサイズ・アクノリッジ	$\overline{\mathrm{DSACK0}}$/$\overline{\mathrm{DSACK1}}$	入力	“L”	–
同期ターミネーション	$\overline{\mathrm{STERM}}$	入力	“L”	–
キャッシュ・インヒビット・イン	$\overline{\mathrm{CIIN}}$	入力	“L”	–
キャッシュ・インヒビット・アウト	$\overline{\mathrm{CIOUT}}$	出力	“L”	Yes
キャッシュ・バースト要求	$\overline{\mathrm{CBREQ}}$	出力	“L”	Yes
キャッシュ・バースト・アクノリッジ	$\overline{\mathrm{CBACK}}$	入力	“L”	–
割込み優先レベル	$\overline{\mathrm{IPL0}}$-$\overline{\mathrm{IPL2}}$	入力	“L”	–
割込みペンディング	$\overline{\mathrm{IPEND}}$	出力	“L”	No
オートベクタ	$\overline{\mathrm{AVEC}}$	入力	“L”	–
バス要求	$\overline{\mathrm{BR}}$	入力	“L”	–
バス許可	$\overline{\mathrm{BG}}$	出力	“L”	No
バス許可アクノリッジ	$\overline{\mathrm{BGACK}}$	入力	“L”	–
リセット	$\overline{\mathrm{RESET}}$	入出力	“L”	No
ホルト	$\overline{\mathrm{HALT}}$	入力	“L”	–
バス・エラー	$\overline{\mathrm{BERR}}$	入力	“L”	–
キャッシュ・ディセーブル	$\overline{\mathrm{CDIS}}$	入力	“L”	–
MMUディセーブル	$\overline{\mathrm{MMUDIS}}$	入力	“L”	–
パイプの再充てん	$\overline{\mathrm{REFILL}}$	出力	“L”	No
マイクロシーケンサ・ステータス	$\overline{\mathrm{STATUS}}$	出力	“L”	No
クロック	CLK	入力	–	–
電源	V_{CC}	入力	–	–
グランド	GND	入力	–	–

第 6 章

オンチップ キャッシュ・メモリ

MC68030マイクロプロセッサは、論理(仮想)アドレスでアクセスされる256バイトの命令キャッシュと256バイトのデータ・キャッシュを内蔵しています。これらのキャッシュは、外部バス動作を低減し、命令のスループットを大きくすることによって性能を改善します。

外部バス動作を減らせば、MC68030の性能を低下させることなく、外部デバイスに対するバスの可用性を増大させることができるため、全体的な性能が向上します。プログラムで必要な命令ワードやデータがオンチップ・キャッシュにあるときには、命令のスループットが大きくなり、外部バスを使用してそれらにアクセスする時間は要りません。さらに、命令ワードとデータを同時にアクセスできれば、命令のスループットは一層大きくなります。

図6-1に示すように、命令キャッシュとデータ・キャッシュは、別々のオンチップ・アドレス・バスおよびデータ・バスに接続されています。アドレス・バスを組み合わせて、メモリ管理ユニット(MMU)に論理アドレスが供給されます。MC68030は、MMUのアドレス変換キャッシュにある論理アドレスを変換するためのアクセスを開始すると同時に、要求された命令またはデータ・オペランドに対応するキャッシュへのアクセスを開始します。命令キャッシュまたはデータ・キャッシュでヒットがあり、MMUがライトに対するアクセスを認可すると、情報はキャッシュから(リード時)またはキャッシュおよびバス・コントローラ(ライト時)に適宜転送されます。ヒットしなかった場合、MMUでアドレス変換されたアドレスを外部バス・サイクルに使用して、命令またはオペランドを取得します。要求されたオペランドがオンチップ・キャッシュのいずれかに存在するしないに関係なく、MMUのアドレス変換キャッシュは、外部サイクルが要求される場合、キャッシュのルックアップと並行して論理－物理アドレス変換を実行します。

6.1 オンチップ・キャッシュの構成と操作

次に、MC68030の内部キャッシュの構成と操作について説明します。

両方のオンチップ・キャッシュとも、それぞれ16ラインで構成される256バイトのダイレクト・マップ方式のキャッシュです。各ラインは4つのエントリよりなり、各エントリは4バイト長です。各ラインのタグ・フィールドには、そのラインの各エントリに対する有効ビットがあり、各エントリは個別に置換え可能です。バス・コントローラは適宜、バースト・モード操作を要求して、各ライン全体を置き換えます。キャッシュ制御レジスタ(CACR)は、スーパバイザ・プログラムからアクセスでき、両方のキャッシュの動作を制御します。

システム・ハードウェアは、キャッシュ・ディセーブル($\overline{\text{CDIS}}$)信号をアサートして、両方のキャッシュをディセーブルすることができます。$\overline{\text{CDIS}}$をアサートすると、CACRのイネーブル・ビットの状態には関係なく、両方のキャッシュをディセーブルすることができます。$\overline{\text{CDIS}}$は主にイン・サー

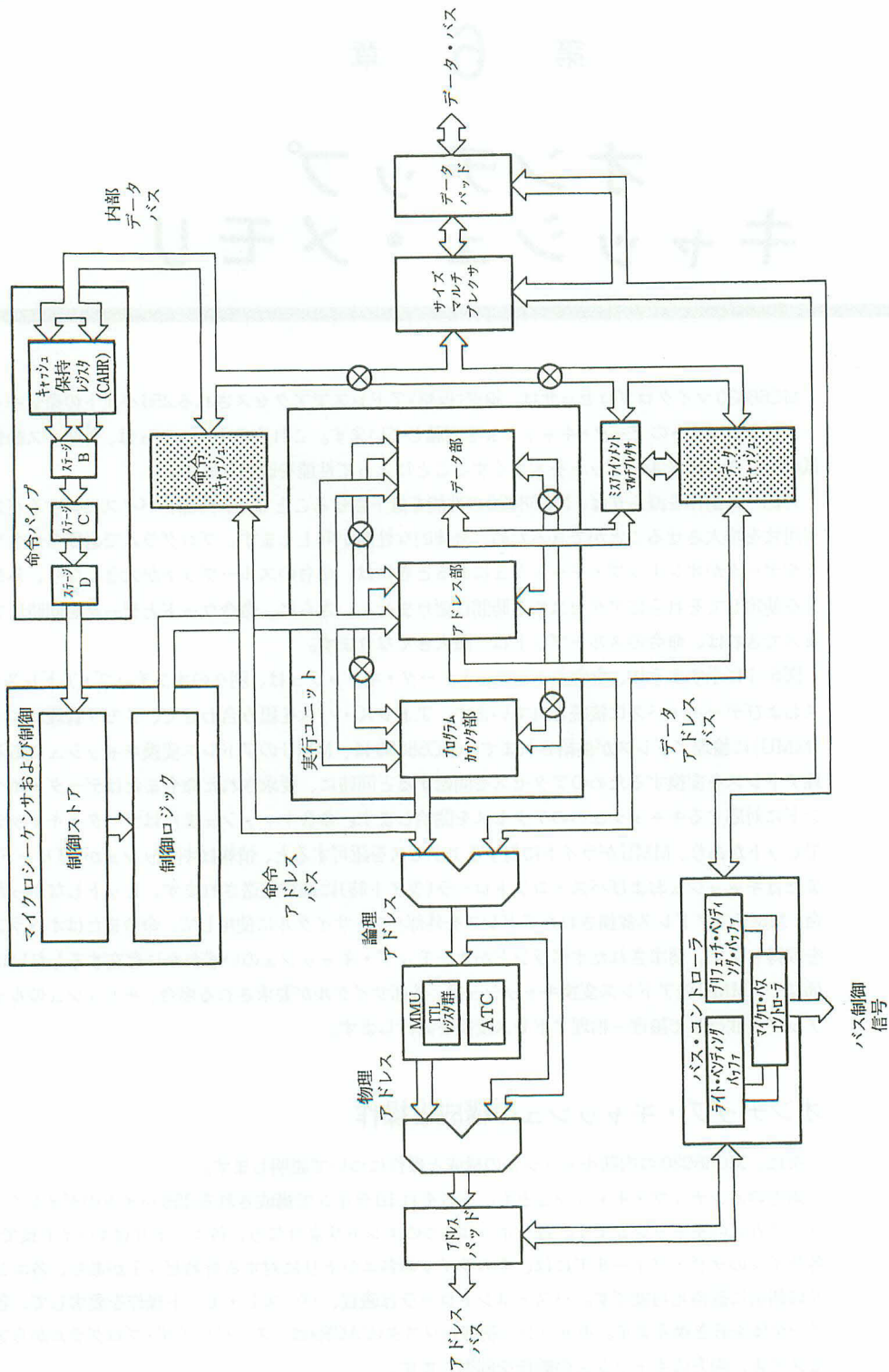



図6-1　内部キャッシュとMC68030

キット・エミュレータで使用するための信号です。

もう1つの入力信号であるキャッシュ・インヒビット・イン($\overline{\text{CIIN}}$)は、バス・サイクルごとに、データ・リードまたは命令フェッチのキャッシングを禁止します。キャッシュしてはならないデータの例としては、I/Oデバイス用データ、および要求されたオペランドのサイズに関係なく、全ポート幅のデータを供給できないメモリ・デバイスからのデータがあります。

以下のパラグラフでは、キャッシュ充てん中の$\overline{\text{CIIN}}$の使用法を説明します。

出力信号であるキャッシュ・インヒビット・アウト($\overline{\text{CIOUT}}$)は、指定された論理アドレスに対応するATCエントリ、またはそのアドレスに対応するトランスペアレント変換レジスタのキャッシュ・インヒビット(CI)ビットの状態を反映します。リードまたはライト・アクセス時に、該当するCIビットがセットされており、かつ外部バス・サイクルが要求されるときは$\overline{\text{CIOUT}}$がアサートされ、そのアクセスに対し命令キャッシュとデータ・キャッシュは無視されます。外部ハードウェアでこの信号を使用して、外部キャッシュへのキャッシングを禁止することもできます。

リード・アクセスが発生し、要求される命令ワードまたはデータ・オペランドが該当するオンチップ・キャッシュに存在している場合(外部バス・サイクルは不要)は、その時点でMMUに無効な変換が存在しないかぎり、MMUは完全に無視されます(次の2つのパラグラフを参照のこと)。したがって、MMUの対応するCIビットの状態も無視されます。MMUは外部バス・サイクルを要求するすべてのアクセスを認可するために使用されます。したがって、アドレス変換が実行可能であり、有効になっていなければなりません。保護が検査され、$\overline{\text{CIOUT}}$信号が適宜アサートされます。

外部アクセスは、次のすべての条件が満たされた場合は、命令キャッシュまたはデータ・キャッシュのいずれかに対して、“キャッシュ可能”と定義されます。

- CACRの該当ビットをセットしてキャッシュがイネーブルされた。
- $\overline{\text{CDIS}}$信号がネゲートされた。
- アクセスに対して$\overline{\text{CIIN}}$信号がネゲートされた。
- アクセスに対して$\overline{\text{CIOUT}}$信号がネゲートされた。
- MMUがアクセスを認可した。

データ・キャッシュと命令キャッシュは、両方とも論理アドレスによって参照されるため、MMUが最初にイネーブルになったときを含め、タスク・スイッチ中または論理アドレスから物理アドレスへのマッピング変更時には、フラッシュしなければなりません。さらに、ページ・ディスクリプタが現在有効とマークされていて、後で無効タイプに変更される場合(コンテクスト・スイッチまたはページ置換え操作のため)は、最初に物理ページに対応するオンチップ命令キャッシュまたはデータ・キャッシュのエントリをクリア(無効化)しなければなりません。そうしないと(無効とマークされたメモリにあるディスクリプタをもつページに対して、オンチップ・キャッシュ・エントリが有効な場合)、プロセッサの動作は予測できません。

同一アドレスへのデータのリードおよびライト・アクセスについても、キャッシュ内のデータが外部メモリと一致するように、キャッシュの可能性に関する一貫したステータスをもっている必要があります。たとえば、ページ内でリード・アクセスを行なうために$\overline{\text{CIOUT}}$がネゲートされ、MMUの構成が変更されて同一ページ内でのライト・アクセスのために、引き続きアサートされた場合、これらのライト・アクセスがキャッシュ内のデータを更新せず、ステール・データとなるおそれがあります。同様に、MMUが複数の論理アドレスを同一物理アドレスにマップしたときは、それらの論理アドレスへの全アクセスが、同じステータスをもっていなければなりません。

6.1.1　命令キャッシュ

命令キャッシュは、図6-2に示すように1ラインのサイズが4ロング・ワード構成になっています。これらの各ロング・ワードは、それぞれが個別に有効ビットをもっているため、独立したキャッシ

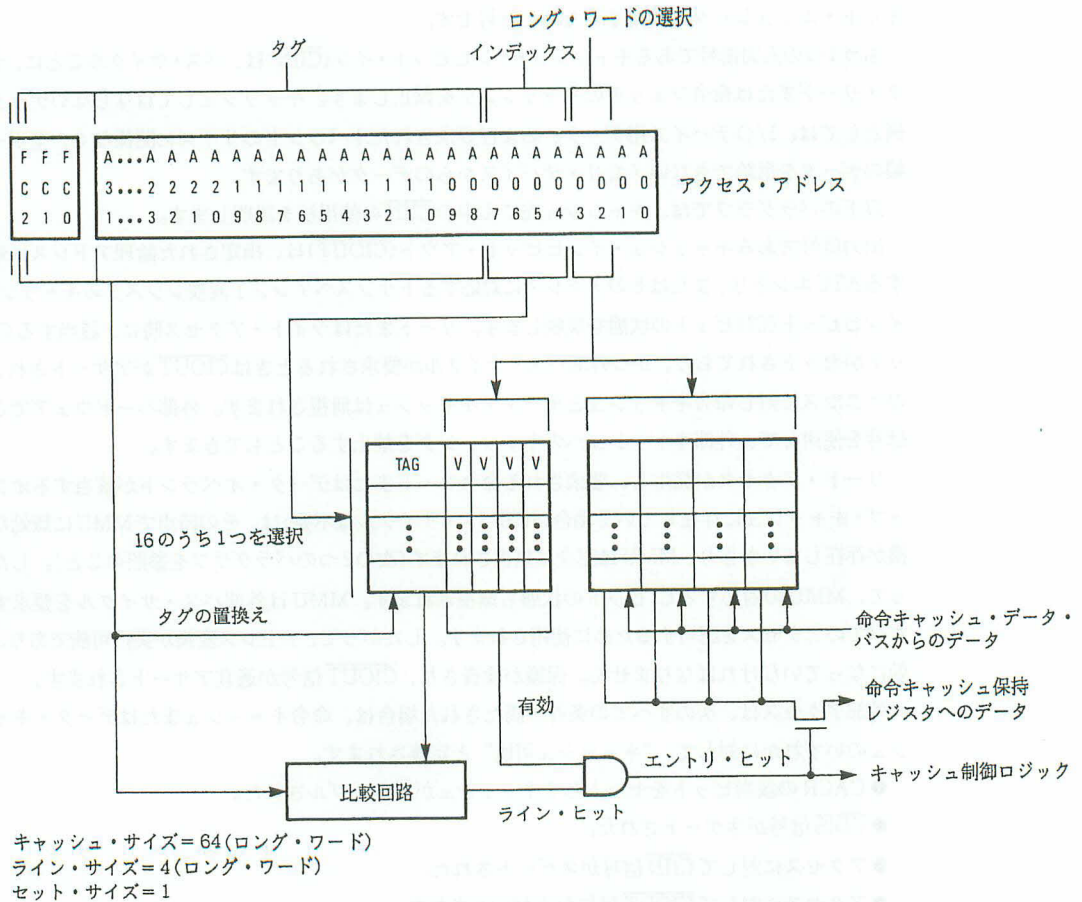



図6-2　オンチップ命令キャッシュの構成

ュ・エントリとみなされます。1ラインにある4つのエントリはすべて、同じタグ・アドレスをもっています。4ロング・ワードすべてに対するバースト充てんは、1ラインの充てんに費やす時間が4ロング・ワードへの非バースト・アクセスに要する同等のバス・サイクル時間と比べて短いときには有利です。なぜなら、後続のアクセスでも、参照したオペランドまたは命令に隣接するかその近傍のメモリ内容へのアクセスが要求される確率が高いためです。高速アクセス・モード(ページ、ニブル、またはスタティック・カラム)をサポートするダイナミックRAMを使用すれば、容易にMC68030のバースト・モードをサポートすることができます。

命令キャッシュはイネーブルされると、命令がCPUに要求されたときに、命令プリフェッチ(命令ワードおよび拡張ワード)を格納するのに使用されます。命令プリフェッチは、プログラム・フローに変更が生じたり(たとえば、分岐など)、ステータス・レジスタを変更する可能性のある命令が実行されるとき(いずれの場合も、命令パイプが自動的にフラッシュされ再充てんされる)を除いて、通常は順次メモリ・アドレスから要求されます。出力信号$\overline{\text{REFILL}}$によってこの状態が通知されます。この信号の操作に関する詳細は、「第12章　アプリケーション情報」を参照してください。

命令キャッシュでは、16ラインのそれぞれに24ビットの最上位論理アドレス・ビットよりなるタグ、FC2ファンクション・コード・ビット(ユーザ・アクセスとスーパバイザ・アクセスを区別するために使用)、および4つの有効ビット(各ロング・ワードに対応)があります。命令キャッシュの構成については、図6-2を参照してください。アドレス・ビットA4－A7で16ラインのうちの1つと関連するタグを選択します。コンパレータは、選択されたタグのアドレスおよびファンクション・

コード・ビットと、アドレス・ビットA8-A31および内部プリフェッチ要求からのFC2を比較し、要求されたワードがキャッシュにあるかどうか判断します。タグが一致し、対応する有効ビット(A2-A3で選択される)がセットされていれば、キャッシュ・ヒットが起こります。キャッシュ・ヒットが起こると、アドレス・ビットA1で選択されるワードが命令パイプに供給されます。

アドレスおよびファンクション・コード・ビットが一致しなかったり、要求されたエントリが有効でない場合はミスが起こります。バス・コントローラは要求された命令ワードに対してロング・ワード・プリフェッチ操作を開始し、キャッシュ・エントリがキャッシュ可能であれば、そのエントリをロードします。キャッシュ・ライン全体を充てんするのに、バースト・モード操作を要求することができます。ファンクション・コードとアドレス・ビットが一致し、対応するロング・ワードが有効でない(ただし、そのラインに対する他の3つの有効ビットの1つまたは複数がセットされている)場合、シングル・エントリ充てん操作により、通常のプリフェッチ・バス・サイクル(単数または複数)(バーストではない)を使用して、要求されたロング・ワードだけを置き換えます。

6. 1. 2 データ・キャッシュ

データ・キャッシュは、PC相対アドレッシング・モードによる参照およびMOVES命令によるアクセスを含め、CPU空間(FC＝＄7)を除く任意のアドレス空間に対してデータ・リファレンスを格納します。データ・キャッシュの操作は、アドレス比較とキャッシュ充てん操作を除いて、命令キャッシュの操作に類似しています。データ・キャッシュの各ラインのタグは、アドレスA8-A31のほかに、ファンクション・コード・ビットFC0、FC1、およびFC2をもっています。キャッシュ制御回路は、ビットA4-A7を使用してタグを選択し、それをアクセス・アドレスの対応するビットと比較して、タグの一致があったかどうかを判断します。アドレス・ビットA2-A3は、キャッシュ内で該当するロング・ワードの有効ビットを選択し、エントリのヒットが起こったかどうか判断します。ミスアラインメントのデータ転送は、2つのデータ・キャッシュ・エントリにまたがることもあります。この場合、プロセッサは一度に1エントリのヒットをチェックします。したがって、アクセスの一部がヒットし、一部がミスするということもあるわけです。ヒットとミスは別々に扱われます。図6-3にデータ・キャッシュの構成を示します。

データ・キャッシュの操作は、リード・サイクルとライト・サイクルでは異なります。データ・リード・サイクルは、命令キャッシュのリード・サイクルとまったく同様に動作し、ミスが発生するとメモリからオペランドを取り出すために、外部サイクルが開始され、アクセスがキャッシュ可能な場合は、データがキャッシュにロードされます。ミスアラインメントのオペランドが2つのキャッシュ・エントリにまたがる場合は、メモリから2つのロング・ワードを取り出す必要があります。また、バースト・モード操作を開始して、データ・キャッシュ全体のラインを充てんすることもできます。CPUアドレス空間からのリード・アクセスおよびアドレス変換テーブル・サーチ・アクセスは、データ・キャッシュには格納されません。

MC68030のデータ・キャッシュは、ライト・スルー・キャッシュです。ライト・サイクルでヒットが発生すると、オペランド・サイズに関係なく、またキャッシュが凍結されている場合でも、データはキャッシュと外部メモリの両方に書き込まれます(MMUがそのアクセスを認可した場合)。MMUがそのアクセスを無効であると判断した場合、そのライトはアボートされ、対応するエントリが無効になり、バス・エラー例外が発生します。キャッシュへのライトは、外部メモリへのライトの前に完了しますので、外部ライトがバス・エラーで終了しても、キャッシュには新しい値が入っています。データ・キャッシュの値は、外部ライト・サイクルが完了する前に別の命令で使用することもできます。ただし、これによって不都合な結果を生じないようにすることが必要です。バスの同期化についての詳細は、「7. 6　バスの同期化」を参照してください。

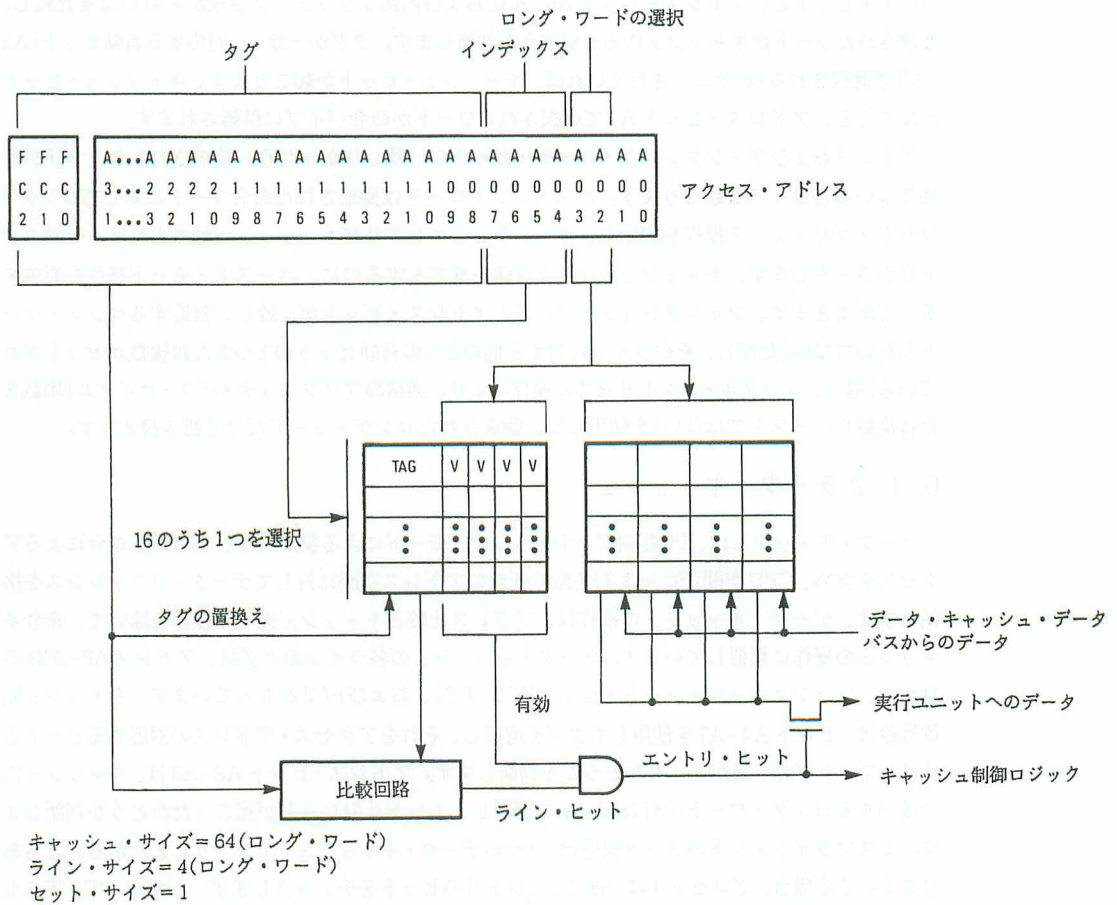



図6-3　オンチップ・データ・キャッシュの構成

6.1.2.1　ライト・アロケーション

スーパバイザ・プログラムは、ライト・サイクルでミスしたデータ・エントリに対し、データ・キャッシュを2種類のアロケーションのいずれかで構成しておくことができます。キャッシュ制御レジスタ内のライト・アロケーション(WA)ビットの状態により、“ノー・ライト・アロケーション”またはライトにおいてキャッシュのデータ・エントリの部分有効化を行なう“ライト・アロケーション”のいずれかを指定します。

“ノー・ライト・アロケーション”(WA＝0)を選択したときは、ライト・サイクルでミスが起こると、データ・キャッシュの内容は変更されません。このモードでは、プロセッサはライト操作中にキャッシュのエントリを置き換えません。キャッシュはライト・ヒットのときにだけ更新されます。

“ライト・アロケーション”(WA＝1)を選択したときは、プロセッサはキャッシュ可能なライト・サイクルで常にデータ・キャッシュを更新しますが、ヒットした更新エントリまたはロング・ワードに整列したロング・ワード・データで更新されたエントリしか有効にしません。ロング・ワードに整列したロング・ワード・データのライトでタグ・ミスが発生したときには、対応するタグが置き換えられ、書込み中のロング・ワードだけが有効としてマークされます。整列していないロング・ワード・データのライト、またはバイトあるいはワード・ライトでタグ・ミスが発生したときには、キャッシュ・ラインの他の3つのエントリだけが無効となり、そのデータはキャッシュに書き込まれず、

タグは変更されず、有効ビットがクリアされます。したがって、整列したロング・ワード・データのライトは、以前の有効データを置き換えることができますが、ミスアラインメントのデータ・ライトまたは非ロング・ワード・データのライトは、以前の有効エントリを無効にすることがあります。

ライト・アロケーションは、次の2つの例外的な状況のいずれかによって起こりうるキャッシュ内でのデータのステールを防止します。

1) 同一タスク内で2つまたはそれ以上の論理アドレスを1つの物理アドレスにマッピングする多重マッピング
2) 同じ物理ロケーションをスーパバイザおよびユーザ・モード・サイクルの両方でアクセス可能にした

データのステールは、“ノー・ライト・アロケーション”モードで操作を行なっていて、次のすべての条件が満足されたときに起こる可能性があります。

- オペレーティング・システムで多重マッピング(オブジェクト別名)が許可されている。
- リード・サイクルが“別名が付けられた”物理アドレスの値をデータ・キャッシュにロードする。
- ライト・サイクルが発生し、上記と同じ別名が付けられた物理オブジェクトを、異なる論理アドレスを使用して参照したためキャッシュ・ミスが発生し、そのキャッシュ(同じページ・オフセットをもつ)が更新されなかった。

図6-4　ノー・ライト・アロケートおよびライト・アロケート・モードの例

●次に、物理オブジェクトが最初の別名を使用してリードを行なうと、キャッシュからステール・データが供給される。

この場合、キャッシュ内のデータはもはや物理メモリ内のデータとは一致しないため、ステール・データとなります。ライト・アロケーション・モードでは、ライト・サイクル中にキャッシュを更新するため、キャッシュ内のデータは物理メモリのデータと一致します。$\overline{\text{CIOUT}}$がアサートされたときには、ライト・サイクルがライト・アロケーション・モードで動作している場合でも、データのキャッシュは完全に無視されます。また、ライト・サイクルでは$\overline{\text{CIIN}}$信号は無視されるため、ライト・アロケーション・モードで動作しているときには、キャッシュ不可能データに対してもキャッシュ・エントリを生成することができます($\overline{\text{CIIN}}$がライトでアサートされるとき)。図6-4に、各モードが5つの異なる状況のもとで動作する様子を示します。

6. 1. 2. 2　リード・モディファイ・ライト・アクセス

リード・モディファイ・ライト・サイクルのリード部分は、データ・キャッシュでは常にミスを起こします。ただし、システムがリード・モディファイ・ライト・サイクル・オペランドの内部キャッシングを許容している場合($\overline{\text{CIOUT}}$および$\overline{\text{CIIN}}$の両方がネゲートされている)、プロセッサはメモリからのデータ・リードを使用してデータ・キャッシュ内のマッチング・エントリを更新するか、マッチング・エントリがない場合にはリード・データにより新しいエントリを生成します。リード・モディファイ・ライト操作のライト部分でも、データ・キャッシュ内のマッチング・エントリの更新が行なわれます。ライト時にキャッシュ・ミスが起こった場合、書込み中のデータに対する新しいキャッシュ・エントリのアロケーションはWAビットで制御されます。しかし、データ・キャッシュはテーブル・サーチ・アクセスを完全に無視しますので、テーブル・サーチ・アクセスでは更新されません。

6. 1. 3　キャッシュの充てん

バス・コントローラは次の2つの方法のいずれかにより、どちらかのキャッシュにロードを行なうことができます。

●シングル・エントリ・モード
●バースト充てんモード

シングル・エントリ・モードでは、バス・コントローラはキャッシュ・ラインのロング・ワード・エントリを1つだけロードします。バースト充てんモードでは、1ライン全体(4つのロング・ワード)を充てんすることができます。これらのモードに必要なバス・サイクルに関する詳細は、「第7章　バス操作」を参照してください。

6. 1. 3. 1　シングル・エントリ・モード

キャッシュ可能なアクセスが開始され、MC68030がバースト・モード操作を要求していないとき、あるいはそれが外部ハードウェアでサポートされていないときは、バス・コントローラが対応するキャッシュ・エントリにロング・ワードを1つ転送します。ロング・ワード全体が要求されます。応答デバイスのポート・サイズが32ビットより小さい場合、MC68030はロング・ワードの充てんに必要な全バス・サイクルを実行します。

転送サイズに関係なく、デバイスがデータ全体のポート幅を供給できないときには、応答デバイスが一貫してキャッシュ・インヒビット・インプット($\overline{\text{CIIN}}$)信号をアサートしなければなりません。たとえば、32ビット・ポートは8ビットまたは16ビット転送であっても、常に32ビットを供給しなければなりません。また、16ビット・ポートは8ビット転送であっても16ビットを転送しなければならないのです。MC68030は、バス・サイクルに対する32ビットのターミネーション信号は、16

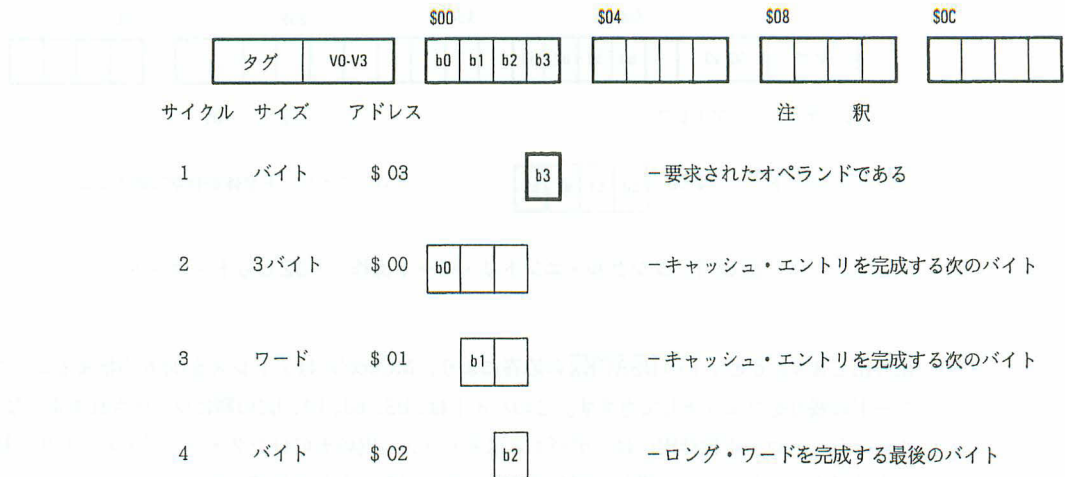



図6-5　シングル・エントリ・モード操作――8ビット・ポート

ビットまたは8ビットしか要求されない場合でも、32の有効データ・ビットの可用性を示すものと想定します。同様に、プロセッサは16ビットのターミネーション信号は16ビットのすべてが有効であることを示すものと想定します。デバイスがデータの全ポート幅を供給できない場合は、キャッシュ・エントリに対応するすべてのバス・サイクルで$\overline{\mathrm{CIIN}}$をアサートしなければなりません。

キャッシュ可能なリード・サイクルが$\overline{\mathrm{CIIN}}$と$\overline{\mathrm{BERR}}$をネゲートしてデータを供給すると、MC68030はキャッシュ・エントリの充てんを試みます。図6-5にキャッシュ内のデータ・ラインの構成を示します。b0、b1、b2などの記号でライン内のバイトを識別します。ラインの各エントリでは、関連タグの有効ビットがロードするロング・ワード・エントリに対応します。1つの有効ビットがロング・ワード全体に適用されますので、シングル・エントリ・モード動作では、完全な32ビット・データを供給しなければなりません。幅が32ビット未満のポートでは、各エントリに対し何回かのリード・サイクルが必要です。

図6-5にバイト・アドレス$ 03から始まる8ビット・ポートからのバイト・データ・オペランドのリード・サイクル例を示します。データ・アイテムがキャッシュ可能な場合、この操作は4バス・サイクルで完了します。MC68030から要求されると、第1サイクルはアドレス$ 03から1バイトを

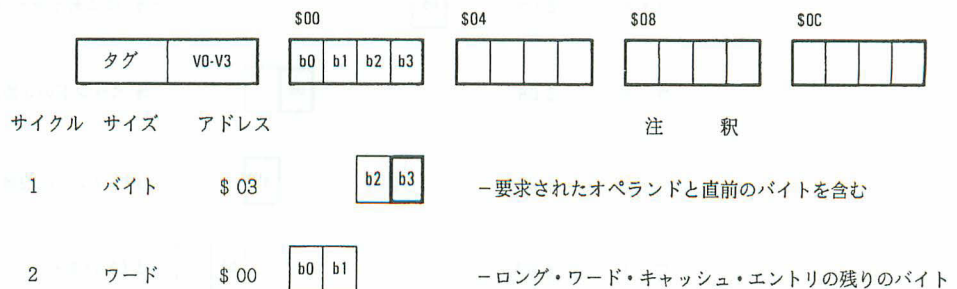



図6-6　シングル・エントリ・モード操作――16ビット・ポート

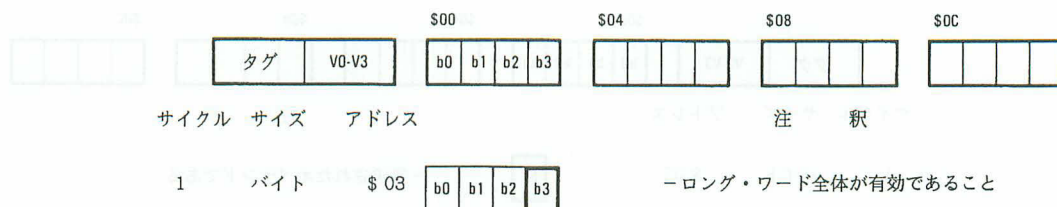



図6-7　シングル・エントリ・モード操作――32ビット・ポート

読み出します。8ビットの$\overline{\text{DSACKx}}$の応答により、MC68030はアドレス$ 00から始まるロング・ワードの残りをフェッチしてきます。このバイトは、b3、b0、b1、b2の順にラッチされます。なお、キャッシュ・ロード操作中には、デバイスはキャッシュ内のそのロング・ワード・エントリに対する全サイクルにおいて、一貫して同じポート・サイズを示さなければなりません。

図6-6に16ビット・ポートからのバイト・データ・オペランドのアクセスを示します。この操作は2つのリード・サイクルを必要とします。最初のサイクルは、アドレス$ 03にあるバイトを要求します。

デバイスが16ビットの$\overline{\text{DSACKx}}$エンコーディングで応答した場合は、アドレス$ 02にあるワード(要求されたバイトを含む)がMC68030に受け入れられます。第2サイクルは、アドレス$ 00にあ

タグ	V0-V3	$00	$04	$08	$0C
			b4 b5 b6 b7	b8 b9 bA bB	

サイクル	サイズ	アドレス		注　釈
1	ﾛﾝｸﾞ・ﾜｰﾄﾞ	$ 06	b6	ーラッチされたオペランドの最初のバイト
2	3バイト	$ 07	b7	ーオペランドの2番目のバイト
3	ワード	$ 04	b4	ーキャッシュ・エントリを$ 04で充てん
4	バイト	$ 05	b5	ー$ 04にあるキャッシュ・エントリの残り
5	ワード	$ 08	b8	ーオペランドの3番目のバイト
6	バイト	$ 09	b9	ーオペランドの最後のバイト
7	ワード	$ 0A	bA	ー$ 08のキャッシュ・エントリを充てん
8	バイト	$ 0B	bB	ー$ 08にあるキャッシュ・エントリの残り

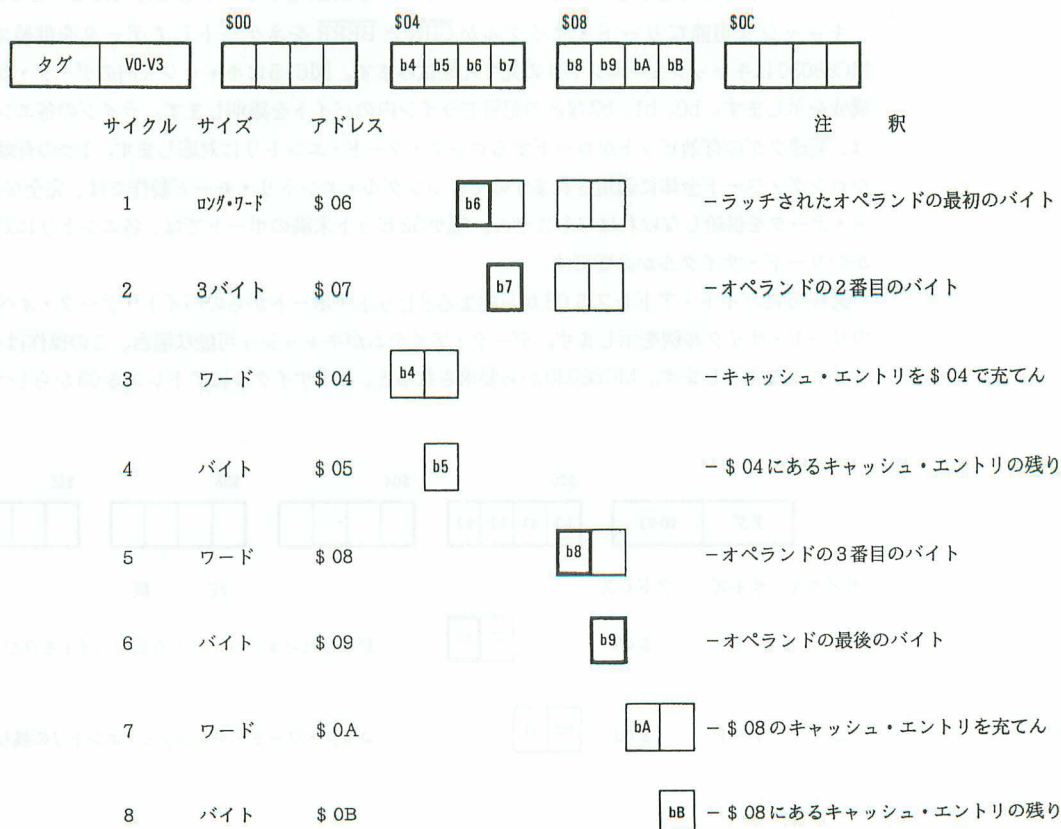



図6-8　シングル・エントリ・モード操作――ミスアラインメントのロング・ワードおよび8ビット・ポート

るワードを要求します。デバイスは再び16ビット$\overline{\text{DSACKx}}$エンコーディングを応答するため、ロング・ワードの残りの2バイトがラッチされ、キャッシュ・エントリが充てんされます。

32ビット・ポートでの、同じ操作を図6-7に示します。リード・サイクルが1サイクルしか必要ありません。このサイクルで4バイトすべて(要求されたバイトも含む)がラッチされます。

要求されたアクセスがミスアラインメントで2つのキャッシュにまたがる場合、バス・コントローラは両方の関連ロング・ワード・キャッシュ・エントリを充てんしようとします。この一例が、奇数ワード境界でロング・ワードを要求するオペランドです。MC68030は最初にオペランドのイニシャル・バイト(最初のロング・ワードにある)をフェッチし、ついでそのキャッシュ・エントリを充てんするための残りのバイトを要求し(ポート・サイズが32ビット未満のとき)、その後オペランドの残りのバイトおよび対応するロング・ワードを要求して、2番目のキャッシュ・エントリを充てんします。ポート・サイズが32ビットの場合、プロセッサは各キャッシュ・エントリに対して1回ずつ計2回アクセスを実行します。

図6-8に、完了するまでに8バス・サイクルを要する、アドレス＄06にある8ビット・ポートからのロング・ワードのミスアラインメントのアクセスを示します。このロング・ワード・オペランドを読み出すには、8つの全アドレスにアクセスした場合、8ビットのポート・サイズ・エンコーディングが返ってくるため、8リード・サイクルを必要とします。これらのサイクルは、要求されたロング・ワードのある2つのキャッシュ・エントリをフェッチします。最初のサイクルはアドレス＄06にあるロング・ワードを要求し、最初に要求されたバイト(b6)を受け取ります。最初のロング・ワード以降の転送は、b7、b4、b5の順に実行されます。残りの4リード・サイクルは、2番目のキャッシュ・エントリの4バイトを転送します。操作全体のアクセスの順序は、b6、b7、b4、b5、b8、b9、bA、およびbBとなります。

次の図6-9に示す例は、16ビットの$\overline{\text{DSACKx}}$エンコーディングを返すデバイスからのミスアラインメントのロング・ワード・オペランドの読出しです。プロセッサはオペランドの最初の部分として、アドレス＄06にあるワードを受け入れ、アドレス＄04からのワードをキャッシュ・エントリに充てんするように要求します。次にプロセッサは、オペランドの2番目の部分として、アドレス＄08にあるワードを読み出し、それもキャッシュに入れます。最後に、プロセッサは＄0Aにあるワードにアクセスして2番目のロング・ワード・キャッシュ・エントリを充てんします。

32ビットの$\overline{\text{DSACKx}}$エンコーディングを返すデバイスからのミスアラインメントのロング・ワー

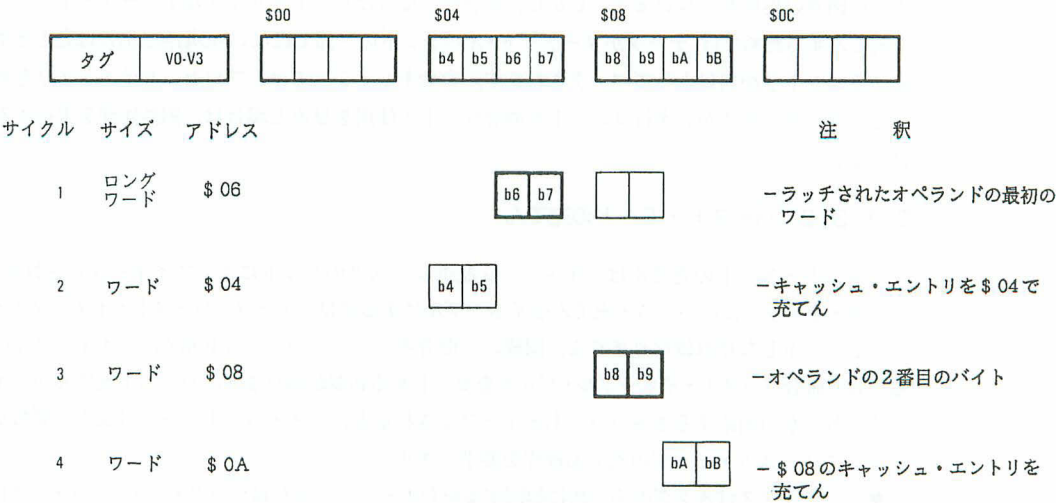



図6-9　シングル・エントリ・モード操作——ミスアラインメントのロング・ワードおよび16ビット・ポート

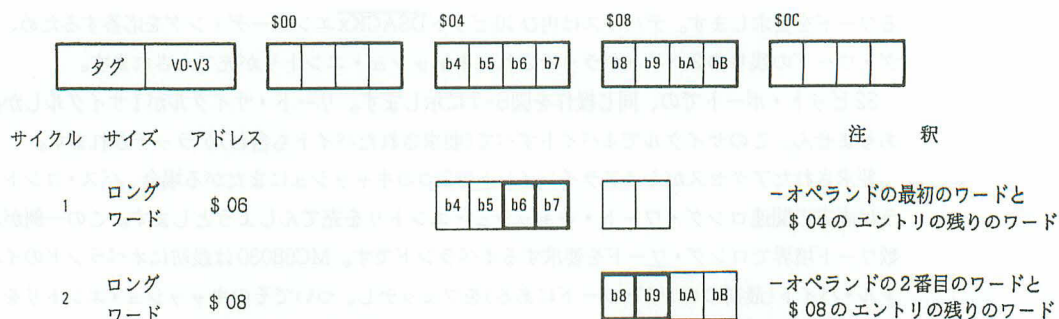



図6-10 シングル・エントリ・モード操作—ミスアラインメントのロング・ワードおよび32ビットの$\overline{\text{DSACKx}}$ポート

ド・オペランドの転送を行なうには、2リード・サイクルが必要です。図6-10に示すとおり、最初のリード・サイクルは、アドレス$ 06にあるロング・ワードを要求し、そのロング・ワードをアドレス$ 04にラッチします。2番目のリード・サイクルは、アドレス$ 08にある2番目のキャッシュ・エントリに対応するロング・ワードをラッチします。$\overline{\text{STERM}}$を使用して32ビット$\overline{\text{DSACKx}}$エンコーディングの代わりに32ビット・ポートを示す場合にも、2リード・サイクルが必要です。

ロング・ワードの全バイトがキャッシュ可能な場合は、エントリを充てんするのに必要な全バス・サイクルをとおして$\overline{\text{CIIN}}$をネゲートしなければなりません。どのバイトもキャッシュ可能でない場合は、対応する全バス・サイクルをとおして$\overline{\text{CIIN}}$をアサートしなければなりません。

$\overline{\text{CIIN}}$信号をアサートすれば、リード・サイクル中にキャッシュが更新されません。ライト・サイクル(リード・モディファイ・ライト・サイクルのライト部分を含む)は、$\overline{\text{CIIN}}$信号がアサートされていてもそれを無視し、キャッシュの状態(ライト・サイクルでのヒットの有無)、CACRのWAビットの状態、およびMMUで指示された状態に応じて、データ・キャッシュの変更を行なうことができます。

キャッシュ・エントリのロードを行なおうとしているときに、バス・エラーが発生すると、エントリ・フィル操作をアボートしますが、必ずしもバス・エラー例外になるとはかぎりません。データ・キャッシュにロードするよう要求されたオペランドの一部(キャッシュ・エントリの残りのバイトではない)に対するリード・サイクルでバス・エラーが発生した場合、プロセッサはすぐにバス・エラー例外の処理を行ないます。しかし、エラーになったリード・サイクルがデータ・キャッシュを充てんするためだけ(データがターゲット・オペランドの一部ではない)の場合、例外は発生せず対応するエントリが無効としてマークされます。命令キャッシュでは、プロセッサはエントリを無効としてマークしますが、実行ユニットが命令ワードの使用を試みた場合は、例外処理を実行するだけです。

6. 1. 3. 2 バースト・モードの充てん

バースト・モードの充てんは、キャッシュ制御レジスタのビットによってイネーブルされます。データ・キャッシュのバースト充てんをイネーブルにするには、データ・バースト・イネーブル・ビットをセットしなければなりません。同様に、命令キャッシュのバースト充てんをイネーブルにするには、命令バースト・イネーブル・ビットをセットする必要があります。バースト充てんがイネーブルされ、かつ対応するキャッシュもイネーブルされると、バス・コントローラは次のいずれかのケースでバースト・モードの充てん操作を要求します。

- インデックス付きタグの不一致に起因する命令キャッシュまたはデータ・キャッシュ・ミスに対するリード・サイクル

●リード・サイクルでタグが一致するが、ライン内のすべてのロング・ワードが無効

バス・コントローラはキャッシュ・バースト要求信号($\overline{\mathrm{CBREQ}}$)をアサートして、バースト・モードの充てん操作を要求します。応答デバイスはキャッシュ可能な1～4ロング・ワード・データを順次供給するか、ロング・ワードのデータがキャッシュ可能でない場合は、キャッシュ・インヒビット入力信号($\overline{\mathrm{CIIN}}$)をアサートすることができます。応答デバイスがバースト・モードをサポートしておらず、$\overline{\mathrm{STERM}}$でサイクルを終了する場合は、キャッシュ・バースト・アクノリッジ($\overline{\mathrm{CBACK}}$)信号をアサートして、その要求に対して認識応答を行なってはなりません。MC68030は$\overline{\mathrm{DSACKx}}$で終了するサイクル中には、$\overline{\mathrm{CBACK}}$がアサートされても無視します。

キャッシュ・バースト要求信号($\overline{\mathrm{CBREQ}}$)は、参照外部デバイスからのバースト・モード操作を要求します。バースト・モードで操作を行なうには、デバイスまたは外部ハードウェアは必要に応じて、下位アドレス・ビットをインクリメントでき、また現在のサイクルは32ビットの同期転送($\overline{\mathrm{STERM}}$)でなければなりません。これについては、「第7章　バス操作」で説明しています。デバイスはMC68030が$\overline{\mathrm{CBREQ}}$をアサートするサイクルの終わりで、$\overline{\mathrm{CBACK}}$ ($\overline{\mathrm{STERM}}$と同時に)もアサートしなければなりません。$\overline{\mathrm{CBACK}}$により、プロセッサはアドレスおよびバス制御信号をドライブし続け、後続の各サイクルの終了時($\overline{\mathrm{STERM}}$で定義される)に、最高4サイクルの間(4つのロング・ワードを読み込むまで)、次のキャッシュ・エントリのための新しいデータ値をラッチしなければなりません。

キャッシュ・バーストが開始されると、第1サイクルは、実行ユニットによって明示的に要求される命令ワードまたはデータ・アイテムに対応するキャッシュ・エントリのロードを試みます。それ以降のサイクルは、キャッシュ・ラインへの後続エントリのためのものです。オペランドが1つのキャッシュ・ライン内で2つのキャッシュ・エントリにまたがっているときのミスアラインメントの転送の場合、最初のサイクルは下位アドレスにオペランド部分を含むキャッシュ・エントリに対応します。

図6-11に4サイクルのバースト操作を示します。第2、第3、そして第4サイクルは、バースト・モードで実行されます。

バースト操作の第1サイクルと後続サイクルとの違いは、第1サイクルはマイクロシーケンサで要求されたもので、バースト充てんサイクルはバス・コントローラで要求されたものであるためです。したがって、第1サイクルからのデータが返ってくると、すぐにそれを実行ユニット(EU)で利用できます。しかし、バースト充てんサイクルからのデータは、バースト操作が完了するまでEUで利用することはできません。マイクロシーケンサは、ミスアラインメントのデータ・オペランドに対して2つの別々の要求を出しているため、EUは第1サイクルが完了した後、バースト操作中に返されたミスアラインメントのオペランドの最初の部分しか利用できません。マイクロシーケンサは、バースト操作が完了するまで待ってからでないと、オペランドの2番目の部分を要求することができません。通常、2番目の部分への要求は、バースト操作の第2サイクルが異常終了しないかぎり、データ・キャッシュ・ヒットになります。バースト操作メカニズムにより、アドレスはラップアラウンドで

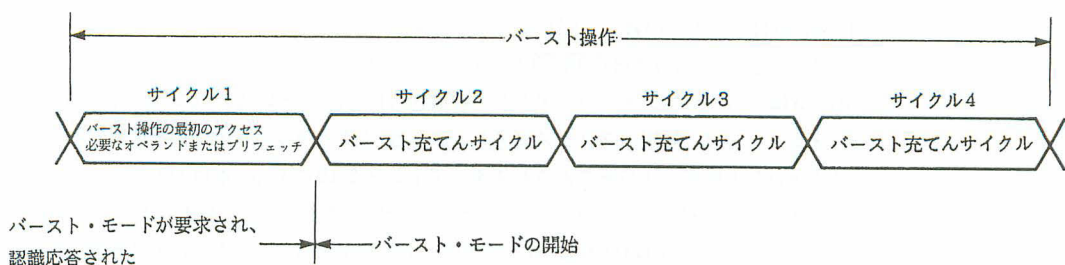



図6-11　バースト操作サイクルとバースト・モード

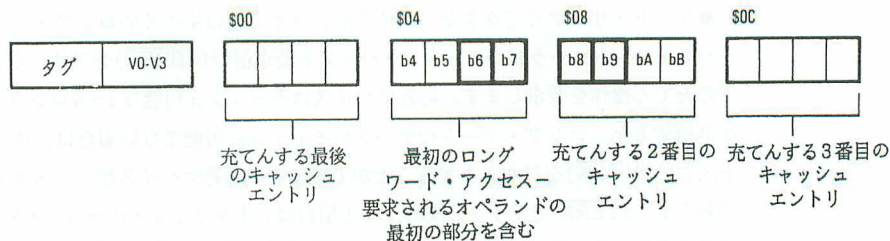



図6-12　バースト充てんのラップアラウンドの例

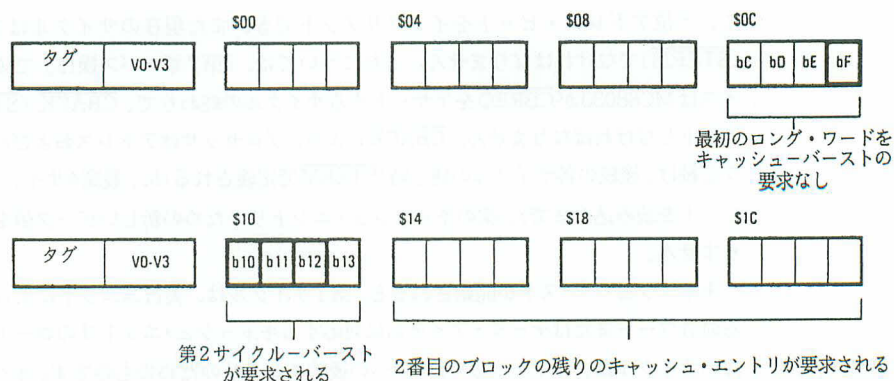



図6-13　遅延バースト充てんの例

きるため、初期アドレスおよびオペランドのアラインメントには関係なく、キャッシュ・ラインの4つのロング・ワードすべてを1回のバースト操作で充てんすることができます。外部メモリ・システムの構造により、アドレス・ビットA2およびA3を外部でインクリメントして、キャッシュにロードするための正しい順序で、ロング・ワードを選択しなければなりません。MC68030はバースト・サイクルの間、アドレス・バス全体を一定に保持します。図6-12にこのアドレスのラップアラウンドを示します。最初のサイクルは、アドレス＄6からのロング・ワード・アクセスです。応答デバイスが$\overline{\text{CBACK}}$と$\overline{\text{STERM}}$(32ビット・ポートを通知)を返すため、ベース・アドレス＄04にあるロング・ワード全体が転送されます。$\overline{\text{CBREQ}}$がアサートされたときの初期アドレスは＄06ですので、次にキャッシュにバースト充てんされるエントリは、アドレス＄08、そして＄0C、最後に＄00に対応していなければなりません。このアドレッシングは既存のニブル・モードのダイナミックRAMと同じであり、A2およびA3に対応する外部モジュロ4カウンタを使用して、ページおよびスタティック・カラム・モードによりサポートすることができます。

MC68030はアクセスの残りが同じキャッシュ・ラインに対応しない場合は、ミスアラインメントのアクセスの最初の部分では$\overline{\text{CBREQ}}$をアサートしません。

図6-13はミスアラインメントのアクセスの最初の部分がアドレス＄0Fにある例を示します。32ビット・ポートでは、最初のアクセスがアドレス＄0Cのキャッシュ・エントリに対応し、シングル・エントリ・ロード操作により充てんされます。アドレス＄10にある2番目のアクセスは、2番目のキャッシュ・ラインに対応しており、バースト充てんを要求するため、プロセッサは$\overline{\text{CBREQ}}$をアサートします。このバースト操作中は、ロング・ワード＄10、＄14、＄18、および＄1Cはすべてこの順序で充てんされます。

プロセッサは、次のいずれかの状態が存在するときには$\overline{\text{CBREQ}}$をアサートしません。

- 該当するキャッシュがイネーブルされていない。
- キャッシュのバースト充てんがイネーブルされていない。
- 該当するキャッシュのキャッシュ凍結ビットがセットされている。
- 現在の操作がリード・モディファイ・ライト操作のリード部分である。
- メモリ管理ユニット(MMU)が現在のページに対しキャッシングを禁止している。
- サイクルが2つのキャッシュ・ラインにまたがる(モジュロ16の境界を越える)オペランドの最初のアクセスに対するものである。

また、$\overline{\text{CIIN}}$および$\overline{\text{BERR}}$のアサート、および$\overline{\text{CBACK}}$の早期ネゲートを行なうと、次のとおりバースト操作に影響を与えます。

バースト操作の第1サイクルで$\overline{\text{CIIN}}$をアサートすると、プロセッサによってデータがラッチされ(オペランド全体が最初のサイクルでラッチされる)、要求されたオペランドが整列している場合、データは命令パイプまたは実行ユニットに渡されます。ただし、データは対応するキャッシュにはロードされません。さらに、MC68030は$\overline{\text{CBREQ}}$をネゲートし、バースト操作がアボートされます。要求されたオペランドの一部がリードされたままのとき(ミスアラインメントのために)には、$\overline{\text{CBREQ}}$をネゲートして、適当なアドレスから第2リード・サイクルを開始されます。

バースト操作の第2、第3、第4サイクルで$\overline{\text{CIIN}}$がアサートされると、そのサイクル中は、該当するキャッシュにデータをロードせずに、$\overline{\text{CBREQ}}$をネゲートして、バースト操作をアボートします。しかし、そのデータが要求されたオペランドの一部を含むサイクルに対するものである場合、実行ユニットはそれを使用します。

バースト操作中に$\overline{\text{CBACK}}$を早期にネゲートしてしまうと、現在のサイクルが通常どおり完了し、正常に転送されたデータが該当するキャッシュにロードされます。ただし、バースト操作はアボートされ、$\overline{\text{CBREQ}}$はネゲートされます。

バースト操作中にバス・エラーが発生しても、バースト操作はアボートされます。バーストの第1サイクル(たとえば、バースト・モードに入る前)でエラーが発生した場合、バスから読み出されたデータは無視され、関連するキャッシュ・ライン全体が“無効”としてマークされます。アクセスがデータ・サイクルの場合、ただちに例外処理が進行します。サイクルが命令フェッチの場合には、バス・エラー例外が保留状態になります。このバス・エラーは、実行ユニットがいずれかの命令ワードを使用しようとした場合にのみ処理されます。パイプライン操作についての詳細は、「11. 2. 2 命令パイプ」を参照してください。

いずれのキャッシュでも、バースト・モードに入ったあと(つまり、第2サイクル以降)でバス・エラーが発生すると、そのサイクルに対応するキャッシュ・エントリが無効としてマークされますが、プロセッサは例外処理を行ないません(マイクロシーケンサがまだデータを要求していない)。命令キャッシュ・バーストの場合、アボートしたサイクルからのデータは、完全に無視されます。保留された命令プリフェッチは、依然保留されたままであり、順次プロセッサによって実行されます。第2サイクルがミスアラインメントのデータ・オペランドのフェッチの一部であり、かつバス・エラーが発生した場合、プロセッサはバースト操作を終了して、$\overline{\text{CBREQ}}$をネゲートします。いったんバーストが終了すると、マイクロシーケンサは2番目の部分に対するリード・サイクルを要求します。バーストが第2サイクルで異常終了しているため、データ・キャッシュはミスとなり、第2外部サイクルが要求されます。$\overline{\text{BERR}}$が再びアサートされると、MC68030は例外処理を実行します。

バースト操作の第1サイクルのアクセスにおいて、“再試行”($\overline{\text{BERR}}$と$\overline{\text{HALT}}$のアサートによって示す)を指示すると、プロセッサはバス・サイクルを再試行し、再び$\overline{\text{CBREQ}}$をアサートします。しかし、バースト操作の第2、第3、または第4サイクル中に$\overline{\text{BERR}}$と$\overline{\text{HALT}}$を同時にアサートして再試行を指示しても、たとえ要求したオペランドがミスアラインメントになっていようと再試行操作は行なわれません。バースト操作のバースト充てんサイクル中に$\overline{\text{BERR}}$と$\overline{\text{HALT}}$をアサートすると、

第6章 オンチップ・キャッシュ・メモリ

31–14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
000000000000000000	WA	DBE	CD	CED	FD	ED	0	0	0	IBE	CI	CEI	FI	EI

WA　=ライト・アロケート
DBE =データ・バースト・イネーブル
CD　=データ・キャッシュのクリア
CED =データ・キャッシュ内のエントリのクリア
FD　=データ・キャッシュの凍結
ED　=データ・キャッシュのイネーブル
IBE =命令バースト・イネーブル
CI　=命令キャッシュのクリア
CEI =命令キャッシュ内のエントリのクリア
FI　=命令キャッシュの凍結
EI　=命令キャッシュのイネーブル

図6-14　キャッシュ制御レジスタ

バス・エラーとホルト操作が別々に行なわれます。プロセッサは$\overline{\text{HALT}}$がネゲートされるまではホルト状態のままで、$\overline{\text{HALT}}$がネゲートされると、前のパラグラフで説明したとおりにバス・エラーを扱います。

6.2　キャッシュ・リセット

プロセッサのハードウェア・リセットが発生すると、両方のキャッシュのすべての有効ビットがクリアされます。両方のキャッシュのキャッシュ制御レジスタ(CACR)のキャッシュ・イネーブル・ビット、バースト・イネーブル・ビット、およびフリーズ・ビット(図6-14参照)もクリアされ、効果的に両方のキャッシュをディセーブルします。CACRのWAビットもクリアされます。

6.3　キャッシュの制御

キャッシュ・アレイには、MC68030の内部制御回路しか直接アクセスできませんが、スーパバイザ・プログラムでCACRのビットをセットしてキャッシュ操作の制御を行なうことができます。スーパバイザは、クリアするキャッシュ・エントリのアドレスをもつキャッシュ・アドレス・レジスタ(CAAR)にもアクセスできます。

6.3.1　キャッシュ制御レジスタ

図6-14に示すキャッシュ制御レジスタ(CACR)は、32ビットのレジスタで、MOVEC命令で読み書きでき、またリセットで間接的に変更することができます。これらの5ビット(4-0)で命令キャッシュを制御します。他の6ビット(13-8)はデータ・キャッシュを制御します。各キャッシュは個別に制御することができますが、1回のMOVEC命令で、両方のキャッシュに対して同時に操作を実行することができます。たとえば、ビット3と11がセットされているロング・ワードをCACRにロードすると、両方のキャッシュともクリアされます。ビット31-14および7-5は、モトローラが使用するために予約されています。これらは現時点ではゼロで読み出され、書込みを行なうと無視されます。将来的な互換性を維持するために、これらのビットをセットするような書込みは行なわないようにしてください。

6.3.1.1　ライト・アロケート

WAビット(ビット13)をセットすると、ライト・サイクルに対してライト・アロケート・モード(「6.1.2.1　ライト・アロケーション」参照)が選択されます。このビットをクリアすると、ノー・

ライト・アロケート・モードになります。このビットはリセットによりクリアされます。スーパバイザはユーザ・タスクとデータを共有したり、タスクが複数の論理アドレスを1つの物理アドレスにマップするときには、このビットをセットしなければなりません。データ・キャッシュがディセーブルされるか凍結されている場合、WAビットは無視されます。

6. 3. 1. 2　データ・バースト・イネーブル

DBEビット(ビット12)をセットすると、データ・キャッシュのバースト充てんがイネーブルされます。データ・キャッシュのバースト充てんが必要なときに、オペレーティング・システムや他のソフトウェアがこのビットをセットします。DBEビットはリセット操作によってクリアされます。

6. 3. 1. 3　データ・キャッシュのクリア

CDビット(ビット11)をセットすると、データ・キャッシュ内のすべてのエントリがクリアされます。オペレーティング・システムおよび他のソフトウェアは、コンテクスト・スイッチの前に、このビットをセットしてキャッシュからデータをクリアします。プロセッサは、MOVEC命令でCACRのCDビットに1をロードしたときに、データ・キャッシュ内のすべての有効ビットをクリアします。CDビットは常にゼロで読み出されます。

6. 3. 1. 4　データ・キャッシュのエントリのクリア

CEDビット(ビット10)をセットすると、データ・キャッシュの任意のエントリがクリアされます。図6-15に示すキャッシュ・アドレス・レジスタ(CAAR)のインデックス・フィールドは、インデックスおよびアドレスのロング・ワード選択部分に対応し、クリアするエントリを指定します。プロセッサは、EDおよびFDビットの状態には関係なく、MOVEC命令がCACRのCEDビットに1をロードしたときに、エントリに対応する有効ビットをクリアすることによって指定したロング・ワードだけをクリアします。CEDビットは常にゼロで読み出されます。

6. 3. 1. 5　データ・キャッシュの凍結

FDビット(ビット9)をセットするとデータ・キャッシュが凍結されます。FDビットがセットされているときに、データ・キャッシュの読出しまたは書込み中に、ミスが起こるとインデックス付きのエントリは置き換えられません。しかし、データ・キャッシュでライト・サイクルがヒットすると、キャッシュが凍結されている間にもエントリが更新されます。FDビットがクリアされているとき、リード・サイクル中にデータ・キャッシュでミスが起こった場合は、エントリ(またはライン)が充てんされ、ミスしたライトでのエントリの充てんはWAビットで制御されます。FDビットはリセットによりクリアされます。

6. 3. 1. 6　データ・キャッシュのイネーブル

EDビット(ビット8)をセットすると、データ・キャッシュがイネーブルされます。このビットがクリアされると、データ・キャッシュがディセーブルされます。EDビットはリセット操作でクリアされます。スーパバイザは通常データ・キャッシュをイネーブルにしますが、必要に応じてEDをクリアしてシステムのデバッギングやエミュレーションを行なうことができます。データ・キャッシュをディセーブルにしてもエントリはフラッシュされません。再度イネーブルにすれば、以前に有効であったエントリは有効のままですので、引き続き使用できます。

6. 3. 1. 7　命令バースト・イネーブル

IBEビット(ビット4)をセットすると、は命令キャッシュのバースト充てんがイネーブルになりま

す。命令キャッシュのバースト充てんが必要なときに、オペレーティング・システムや他のソフトウェアがこのビットをセットします。IBEビットはリセット操作でクリアされます。

6.3.1.8 命令キャッシュのクリア

CIビット(ビット3)をセットすると、命令キャッシュのすべてのエントリがクリアされます。オペレーティング・システムや他のソフトウェアは、コンテクスト・スイッチの前に、このビットをセットしてキャッシュから命令をクリアします。プロセッサは、MOVEC命令がCACRのCIビットに1をロードしたときに、命令キャッシュ内のすべての有効ビットをクリアします。CIビットは常にゼロで読み出されます。

6.3.1.9 命令キャッシュのエントリのクリア

ビット2、CEIビットをセットすると、命令キャッシュの任意のエントリがクリアされます。図6-15に示すキャッシュ・アドレス・レジスタ(CAAR)のインデックス・フィールドは、インデックスおよびアドレスのロング・ワード選択部分に対応し、クリアするエントリを指定します。プロセッサは、EIおよびFIビットの状態には関係なく、MOVEC命令がCACRのCEIビットに1をロードしたときに、エントリに対する有効ビットをクリアすることにより指定したロング・ワードだけをクリアします。CEIビットは常にゼロで読み出されます。

6.3.1.10 命令キャッシュの凍結

ビット1、FIビットをセットすると命令キャッシュが凍結されます。FIビットがセットされているときに、命令キャッシュでミスが発生するとエントリ(またはライン)は置き換えられません。FIビットがゼロにクリアされているとき、命令キャッシュでミスが発生した場合は、エントリ(またはライン)が充てんされます。FIビットはリセットによりクリアされます。

6.3.1.11 命令キャッシュのイネーブル

ビット0、EIビットをセットすると、命令キャッシュがイネーブルされます。このビットがクリアされると、命令キャッシュがディセーブルされます。EIビットはリセットでクリアされます。スーパバイザは通常命令キャッシュをイネーブルにしますが、必要に応じてEIをクリアしてシステムのデバッギングやエミュレーションを行なうことができます。命令キャッシュをディセーブルにしてもエントリはフラッシュされません。再度イネーブルにすると、以前に有効であったエントリは有効のままですので、引き続き使用することができます。

6.3.2 キャッシュ・アドレス・レジスタ

キャッシュ・アドレス・レジスタ(CAAR)は32ビットのレジスタであり、その構成を図6-15に示します。インデックス・フィールド(ビット7-2)には、“キャッシュ・エントリのクリア”操作のためのアドレスがあります。このフィールドのビットは、アドレスのビット7-2に対応し、キャッシュ・ラインのインデックスおよびロング・ワードを指定します。現在はインデックス・フィールドしか使用されていませんが、このレジスタの全32ビットがすべて実装されており、モトローラが使用するために予約されています。

31 — 8	7 — 2	1 — 0
キャッシュ・ファンクション・アドレス	インデックス	

図6-15 キャッシュ・アドレス・レジスタ

第 7 章

バス操作

本章ではバス、バスを制御する信号、およびデータ転送操作のためのバス・サイクルの機能を述べます。また、エラーおよびホルト状態、バス調停、およびリセット動作についても説明します。バスの操作はプロセッサまたは外部デバイスのどちらがバス・マスタであっても同じです。ここで述べるバス・サイクルの名称と説明は、バス・マスタ側から見たものです。正確なタイミング仕様については、「第13章　電気的特性」を参照してください。

MC68030のアーキテクチャはバイト、ワード、ロング・ワードのオペランドをサポートしていますので、データ転送およびサイズ・アクノリッジ入力($\overline{\mathrm{DSACK0}}$および$\overline{\mathrm{DSACK1}}$)で制御される非同期バス・サイクルを使用して8ビット、16ビット、および32ビットのデータ・ポートにアクセスすることができます。

同期ターミネーション信号($\overline{\mathrm{STERM}}$)で制御される同期バス・サイクルは、32ビット・ポート間のデータ転送だけにしか使用できません。

MC68030は、バイト、ワードおよびロング・ワードのオペランドを任意のバイト境界でメモリに置くことができます。オペランドの境界が揃っていない場合、ポートのサイズに関係なく、オペランド転送には2サイクル以上のバス・サイクルが必要です。32ビット幅以下のポートに対しては、ミスアラインメントがあったり、ポート幅がオペランド・サイズより小さい場合、2サイクル以上のバス・サイクルが必要なこともあります。命令ワードおよび関連する拡張ワードは、ワード境界に整列させておかなければなりません。ワードまたはロング・ワード・オペランドが整列していないと、MC68030はオペランド転送に複数のバス・サイクルを必要とするため、それらをワードまたはロング・ワード境界に整列しておけば、最適なプロセッサ性能が得られることを覚えておいてください。

7.1　バス転送信号

バスはMC68030と外部メモリ、コプロセッサ、または周辺デバイス間で情報の転送を行ないます。外部デバイスは8ビット、16ビット、または32ビットのデータを並列に扱うことができ、かつ本章で述べるハンドシェイク・プロトコルに従わなければなりません。バス転送中に扱うことができる最大ビット数は、ポート幅で定義されます。MC68030は転送のためのアドレスを指定するアドレス・バスとデータを転送するデータ・バスを内蔵しています。制御信号はサイクルの開始、アドレス空間および転送サイズ、そしてサイクルの種類を示します。選択されたデバイスは、サイクルを終了させるために使用する信号によって、サイクルの長さを制御します。ストローブ信号は、1つがアドレス・バス、もう1つがデータ・バスに対応しており、アドレスの有効性を通知するとともにデータのタイミング情報を提供します。

MC68030のバスは、どのポート・サイズでもMC68020のバスと同じ非同期モードで動作します。

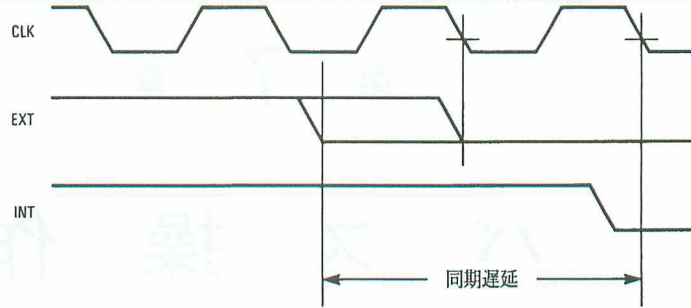



図7-1　外部信号と内部信号の関係

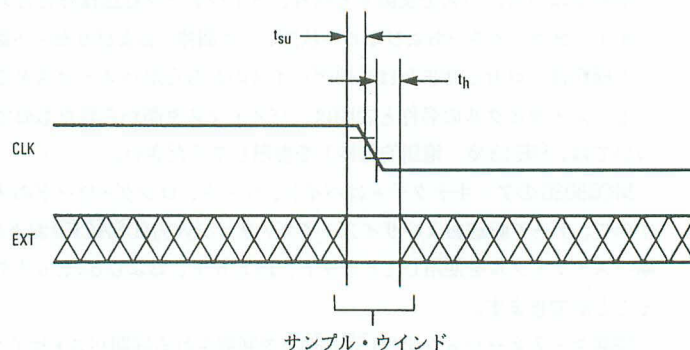



図7-2　非同期入力サンプル・ウインド

非同期動作に使用するバスおよび制御入力信号は、内部でMC68030のクロックに同期しており遅延を生じます。この遅延はMC68030が非同期入力信号をサンプルし、入力をプロセッサの内部クロックに同期させ、さらにそれが“H”か“L”かを判断するするために必要な時間です。

図7-1にクロック信号、代表的な非同期入力、および関連する内部信号を示します。

さらに、すべての非同期入力に対し、プロセッサはクロック信号の立下りエッジ付近のサンプル・ウインドの中で入力レベルをラッチします。このウインドを図7-2に示します。あるクロックの立下りエッジで入力信号を認識するには、サンプル・ウインドの中では入力が安定していなければなりません。ウインド期間中に入力が変化した場合、プロセッサがどのレベルを認識するかは予測できません。しかし、プロセッサは常にラッチされたレベルを“H”または“L”に決めてからそれを使用します。確実な動作を行なうために、入力セットアップ時間およびホールド時間に適合することはもとより、どの入力信号も本章で述べるプロトコルに従わなければなりません。

32ビットのポート・サイズをもつデバイスは同期モードの転送を行なうことができます。同期動作では、入力信号は外部的にプロセッサ・クロックと同期しており、同期化のための遅延は発生しません。

同期入力($\overline{\text{STERM}}$, $\overline{\text{CBACK}}$, および$\overline{\text{CIIN}}$)は、適切な動作を保証するために、これらがいつアサートまたはネゲートされたかに関係なく、アドレス・ストローブ($\overline{\text{AS}}$)がアサートされている間、バス・サイクル中のクロックのすべての立上りエッジに対し、サンプル・ウインド内で安定していなければなりません。

このサンプル・ウインドは、「第13章　電気的特性」の同期入力セットアップおよびホールド時間で定義されています。

7.1.1　バス制御信号

外部サイクル・スタート($\overline{ECS}$)は、プロセッサがバス・サイクルを開始したことを最も早く示す信号です。MC68030はアドレス、サイズ、ファンクション・コード、リード/ライト、およびキャッシュ・インヒビット・アウト出力をドライブし、$\overline{ECS}$をアサートしてバス・サイクルを開始します。しかし、プロセッサが必要なプログラムまたはデータ・アイテムをオンチップ・キャッシュで見つけるか、メモリ管理ユニット(MMU)のアドレス変換キャッシュ(ATC)でミスが発生する、あるいはMMUがアクセス時にフォールトを検出すると、プロセッサはアドレス・ストローブ($\overline{AS}$)をアサートせずに、そのサイクルをアボートします。$\overline{ECS}$は最終的に$\overline{AS}$で条件付けられる各種タイミングを発生するのに使用できます。内部キャッシュのヒット、ATCのミス、またはMMUフォールトが発生した場合に$\overline{ECS}$がアサートされたあと、バス・サイクルがアボートするおそれがあるため、ASによる条件付けが必要になることがあります。$\overline{AS}$のアサートは、これらの内部状態でバス・サイクルがアボートされなかったことを示します。

オペランド転送の最初の外部バス・サイクル中に、$\overline{ECS}$によってオペランド・サイクルのスタート($\overline{OCS}$)信号がアサートされます。オペランド全体を転送するのに数サイクルのバス・サイクルが必要なときは、最初の外部バス・サイクルの初めにだけ、$\overline{OCS}$がアサートされます。$\overline{OCS}$に関するかぎり、“オペランド” はプログラムまたはデータ・アイテムで、実行ユニットが要求する1つのアイテムのことです。

ファンクション・コード信号(FC0～FC2)もバス・サイクルの初めにドライブされます。これらの3つの信号は、アドレスが適用される8つのアドレス空間(表4-1参照)の1つを選択します。現在5つのアドレス空間が定義されています。残りの3つのうち1つはユーザ定義のために予約されており、2つは将来の使用に備えてモトローラが予約しています。ファンクション・コード信号は、$\overline{AS}$がアサートされている間有効です。

バス・サイクルの初めに、サイズ信号(SIZ0、SIZ1)が$\overline{ECS}$およびファンクション・コード信号とともにドライブされます。SIZ0とSIZ1はオペランド・サイクル(1つまたは複数のバス・サイクルよりなる)、または32ビット未満のポート・サイズをもつデバイスからのキャッシュ充てん操作中に、転送する残りのバイト数を示します。表7-2にSIZ0とSIZ1のエンコーディングを示します。これらの信号は$\overline{AS}$がアサートされている間有効です。

リード/ライト(R/$\overline{W}$)信号は、バス・サイクル中の転送の方向を決めます。この信号は、バス・サイクルの初めに状態を変え、$\overline{AS}$がアサートされている間有効です。R/$\overline{W}$はライト・サイクルの前にリード・サイクルがあるか、またはその逆の場合にだけ状態を変えます。この信号は2つのライト・サイクルの間連続して、“L” になったままのこともあります。

リード・モディファイ・ライト・サイクル信号($\overline{RMC}$)は、リード・モディファイ・ライト操作の最初のバス・サイクルの初めにアサートされ、その操作の最終バス・サイクルが完了するまでアサートされたままになっています。$\overline{RMC}$信号はリード・モディファイ・ライト操作の後に続くバス・サイクルのステート0が終わる前にネゲートされることが保証されています。

7.1.2　アドレス・バス

アドレス・バス信号(A0～A31)は、バス・サイクル中に転送するバイトのアドレス(または、最上位バイト)を定義します。プロセッサはバス・サイクルの初めにバス上にアドレスを置きます。アドレスは$\overline{AS}$がアサートされている間有効です。

7.1.3　アドレス・ストローブ

アドレス・ストローブ($\overline{AS}$)は、アドレス・バス上のアドレスおよび各種制御信号の有効性を示す

タイミング信号です。この信号はバス・サイクルが開始された半クロック・サイクル後にアサートされます。

7. 1. 4　データ・バス

データ・バス信号(D0～D31)は、プロセッサ間で転送されるデータを含む双方向の非多重化パラレル・バスを構成します。リードまたはライト操作により、1回のバス・サイクルで8、16、24、または32ビット(1、2、3、または4バイト)のデータを転送することができます。リード・サイクル中は、データはそのバス・サイクルのクロックの最後の立下りエッジで、プロセッサによってラッチされます。ライト・サイクルでは、ポート幅やオペランド・サイズに関係なくデータ・バスの全32ビットがドライブされます。プロセッサは、ライト・サイクルで$\overline{AS}$がアサートされた半クロック・サイクル後にデータをデータ・バスに置きます。

7. 1. 5　データ・ストローブ

データ・ストローブ($\overline{DS}$)は、データ・バスに印加されるタイミング信号です。リード・サイクルでは、プロセッサは$\overline{DS}$をアサートして外部デバイスにバス上にデータを置くよう通知します。リード・サイクル中は、$\overline{AS}$と同時にアサートされます。ライト・サイクルでは、$\overline{DS}$は書き込むデータがデータ・バス上で有効であることを外部デバイスに知らせます。プロセッサはライト・サイクル中は、$\overline{AS}$がアサートされた1クロック・サイクル後に$\overline{DS}$をアサートします。

7. 1. 6　データ・バッファ・イネーブル

データ・バッファ・イネーブル信号($\overline{DBEN}$)を使用して、データ・バス上にデータが存在している間に外部データ・バッファをイネーブルすることができます。リード操作では、バス・サイクルが開始された1クロック・サイクル後にアサートされ、$\overline{DS}$がネゲートされるとネゲートされます。ライト操作では、$\overline{DBEN}$は$\overline{AS}$がアサートされるとアサートされ、そのサイクルの間アクティブに保持されます。2クロック・バス・サイクルをサポートする同期システムでは、$\overline{DBEN}$のタイミングを使用できない場合があります。

7. 1. 7　バス・サイクル・ターミネーション信号

非同期バス・サイクルでは、外部デバイスはバス・プロトコルの一部としてデータ転送およびサイズ・アクノリッジ信号($\overline{DSACK0}$および$\overline{DSACK1}$、あるいはそのいずれか)をアサートします。リード・サイクルでは、これらの信号によってプロセッサにバス・サイクルを終了してデータをラッチするように通知します。ライト・サイクルでは、これらの信号は外部デバイスが正常にデータを格納し、サイクルを終了してもよいことを知らせます。また、これらの信号は表7-1に示すとおり、プロセッサに直前に完了したバス・サイクルのポート・サイズを知らせます。$\overline{DSACK0}$と$\overline{DSACK1}$のタイミング関係については、「7. 3. 1　非同期リード・サイクル」を参照してください。

同期バス・サイクルでは、外部デバイスがバス・プロトコルの一部として同期ターミネーション信号($\overline{STERM}$)をアサートします。リード・サイクルでは、$\overline{STERM}$がアサートされると、プロセッサがデータをラッチします。ライト・サイクルでは、外部デバイスが正常にデータをラッチしたことを知らせます。いずれの場合も、この信号はサイクルを終了し、32ビット・ポートに対して転送が行なわれたことを示します。$\overline{STERM}$のタイミング関係については、「7. 3. 4　同期リード・サイクル」を参照してください。

バス・エラー($\overline{BEER}$)信号もバス・サイクルの終了を知らせる信号であり、$\overline{DSACKx}$または$\overline{STERM}$がないときに、バス・エラー状態を示すのに使用できます。また、本章および「第13章　電気的特性」で述べる所定のタイミングに適合していれば、$\overline{DSACKx}$または$\overline{STERM}$とともにアサートし

て、バス・エラー状態を表示することができます。また、BERRおよびHALT信号をいっしょにアサートして、再試行動作を示すこともできます。前述したとおり、DSACKxまたはSTERM信号の代わりにあるいはそれらと連係させて、BERRとHALT信号を同時にアサートすることができます。

最後に、オートベクタ(AVEC)信号を使用すれば、割込みアクノリッジ・サイクルを終了して、MC68030に内部でベクタ番号を生成し割込みハンドラ・ルーチンに飛ぶよう指示することができます。AVECは他のバス・サイクルではすべて無視されます。

7.2　データ転送のメカニズム

MC68030のアーキテクチャはバイト、ワード、ロング・ワードのオペランドをサポートしており、データ転送およびサイズ・アクノリッジ入力(DSACK0およびDSACK1で制御される非同期サイクルを使用して、8、16、および32ビットのデータ・ポートにアクセスすることができます。また、STERMで終了する32ビット・ポート間の同期バス・サイクルもサポートしています。バイト、ワードおよびロング・ワードのオペランドを任意のバイト境界に置くことができます。ただし、ポート・サイズに関係なく、オペランドのミスアラインメントの転送には、余分なバス・サイクルが必要になることがあります。

プロセッサがバースト・モードの充てん操作を要求するときは、キャッシュ・バースト要求(CBREQ)信号をアサートしてオンチップ・キャッシュの1つにある1ラインの4つのエントリを充てんするように試みます。このモードは、ニブル、スタティック・カラム、またはページ・モードのダイナミックRAMとコンパチブルです。バースト充てん操作は、それぞれSTERMで終了する同期バス・サイクルを使用して、4つのロング・ワードをフェッチします。

7.2.1　ダイナミック・バス・サイジング

MC68030は各バス・サイクルで、アドレス指定されたデバイスのポート・サイズをダイナミックに解釈して、8ビット、16ビット、および32ビット・ポート間でオペランドの転送ができるようにします。非同期オペランド転送サイクルでは、スレーブ・デバイスが自分のポート・サイズ(バイト、ワード、またはロング・ワード)を通知し、DSACKx入力を使用してプロセッサにバス・サイクルの完了を知らせます。DSACKxのエンコーディングとアサートの結果については表7-1を参照してください。

たとえば、プロセッサがロング・ワードに整列したアドレスからロング・ワードのオペランドを読む命令を実行している場合、プロセッサは最初のバス・サイクルで32ビットを読もうとします(ワードまたはバイト・アドレスの場合については、「7.2.2　ミスアラインメントのバス転送」を参照のこと)。ポートが32ビットのバス幅であると応答した場合、MC68030はデータの32ビット全部をラッチし、次の操作に移ります。ポートが16ビットのバス幅であると応答した場合、MC68030は16ビットの有効データをラッチし、別のバス・サイクルを実行してもう一方の16ビット・データを取得します。

8ビット・ポートの操作も同様ですが、この場合は4リード・サイクルが必要です。アドレス指定されたデバイスは、DSACKx信号を使用してポート幅を知らせます。たとえば、32ビット・デバイスは常に32ビット・ポートのDSACKxを返します(バス・サイクルがバイト、ワード、またはロング・ワード操作かどうかに関係なく)。

ダイナミック・バス・サイジングでは、特定のポート・サイズとの間で転送に使用するデータ・バスの部分は固定されていることが必要です。32ビットのポートはデータ・バスのビット31〜0、8ビットのポートはデータ・バスのビット31〜24になければなりません。この条件があるため、8ビット・ポートおよび16ビット・ポートとの間でデータ転送を行なうのに必要なバス・サイクルの回

表7-1　$\overline{DSACK}$コードと結果

$\overline{DSACK1}$	$\overline{DSACK0}$	結　　果
H	H	現行バス・サイクルにウエイト・ステートを挿入
H	L	完了サイクルーデータ・バスのポート・サイズが8ビット
L	H	完了サイクルーデータ・バスのポート・サイズが16ビット
L	L	完了サイクルーデータ・バスのポート・サイズが32ビット

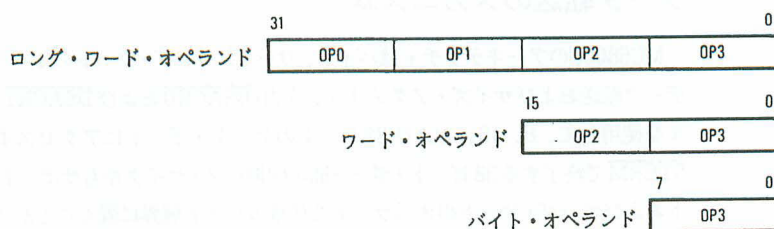



図7-3　オペランドの内部表現

数を最小にして、MC68030が有効なデータを正しく転送できるようにしています。MC68030は、常にすべてのバス・サイクルにおいて最大量のデータを転送しようと試みます。つまり、ロング・ワード操作では、バス・サイクルを開始するとき、ポートが常に32ビット幅であると仮定します。

オペランドのバイトには図7-3に示すような名前が付けられています。ロング・ワード・オペランドの最上位バイトがOP0で、OP3が最下位バイトです。ワード長オペランドの2バイトはOP2(最上位)とOP3です。バイト長オペランドの1バイトはOP3です。以下の図および説明には、これらの名前を用いています。

図7-4に、MC68030のバス上で8、16、および32ビットのデバイスに対して必要なデータ・ポートの構成を示します。図7-4に示す4バイトは、内部データ・バスとデータ・マルチプレクサを通して外部データ・バスに接続されています。この経路を通して、MC68030はダイナミック・バス・サイジングおよびオペランドのミスアラインメントをサポートします。オペランドのミスアラインメントの定義については、「7. 2. 2　ミスアラインメントのバス転送」を参照してください。データ・マルチプレクサは、アドレスとデータ・サイズのさまざまな組合せに対して、必要な接続を確立します。

マルチプレクサは32ビット・バスの4バイトを取り込み、それらを必要な位置に分配します。たとえば、OP0 は通常の場合はD31～D24 に分配されますが、ミスアラインメントの転送を可能にするために、他の任意のバイト位置にも分配できるようになっています。他のオペランド・バイトについても同様です。バイトの位置はサイズ(SIZ0とSIZ1)およびアドレス(A0とA1)出力によって決まります。

SIZ0およびSIZ1の出力は、表7-2に示すように現在のバス・サイクルで転送する残りのバイト数を示します。

ライト・バス・サイクルまたはキャッシュ不可能なリード・バス・サイクルで転送されるバイト数は、SIZ0およびSIZ1出力で示すサイズ以下であり、ポート幅とオペランドのアラインメントによって異なります。たとえば、ロング・ワードをワード・ポートへ転送する場合の最初のバス・サイクルでは、サイズ出力は4バイトを転送すること示しますが、そのバス・サイクルでは2バイトしか転送されません。

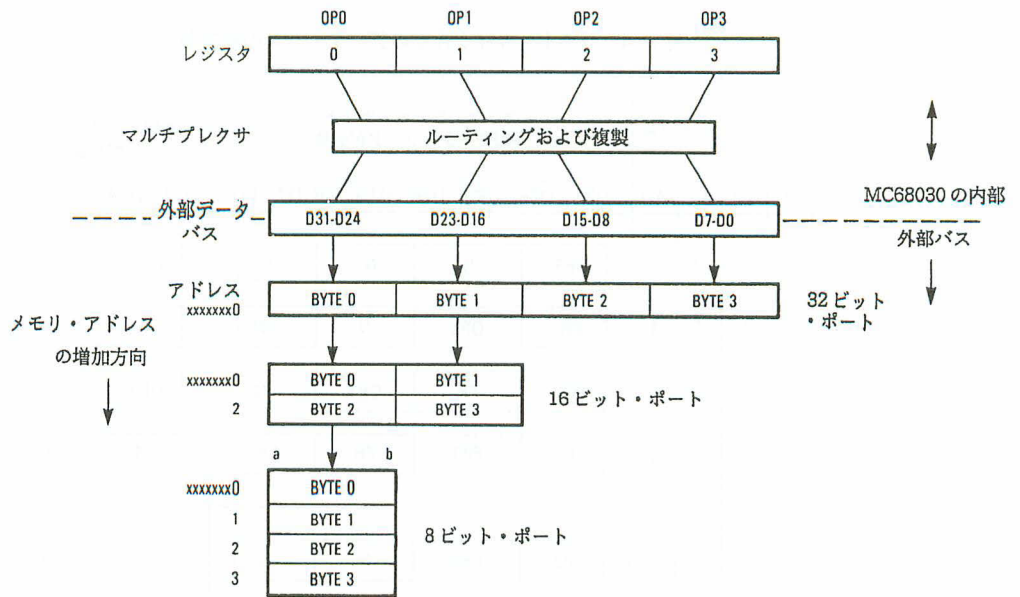



図7-4　MC68030と各種ポート・サイズのインタフェース

キャッシュ可能なリード・サイクルは、常にポート・サイズで示すバイト数を転送しなければなりません。

アドレス・ラインA0およびA1も、データ・マルチプレクサの動作に影響を与えます。オペランド転送中、A2〜A31 はアクセスされるオペランド部分のロング・ワードのベース・アドレスを示し、A0とA1はそのベースからのバイト・オフセットを示します。表7-3に、A0とA1のエンコーディングと対応するロング・ワード・ベースからのバイト・オフセットを示します。

表7-4にキャッシュ可能なリード・サイクルに対して、データ・バスに要求されるバイト数を示します。OPnで示すエントリは、そのバス・サイクル中に読出しまたは書込みが行なわれる要求オペランドの部分であり、バス・サイクルに対してSIZ0、SIZ1、A0、およびA1で定義されています。PRnとNnバイトは、それぞれメモリ内での前および後のバイトに対応しており、内部キャッシュが正しく動作するには指定されたポート・サイズ(ロング・ワードまたはワード)のデータ・バスで有効になっていなければなりません(キャッシュ可能なアクセスでは、MC68030は任意のポート・サイズに対してデータ・バスのすべての部分が有効であると仮定しています)。この表はキャッシュ不可能なリード・サイクルにも適用されます。ただし、PRnとNnのラベルが付いているバイトは必要ありませんので、“Don't Care” に置き換えることができます。

表7-2　サイズ信号のエンコーディング

SIZ1	SIZ0	サイズ
0	1	バイト
1	0	ワード
1	1	3バイト
0	0	ロング・ワード

表7-3　アドレス・オフセットのエンコーディング

A1	A0	オフセット
0	0	+0バイト
0	1	+1バイト
1	0	+2バイト
1	1	+3バイト

表7-4 リード・サイクルでのデータ・バスの要求条件

転送サイズ	サイズ		アドレス		ロング・ワード・ポート外部データ・ワードが必要				ワード・ポート外部データ・ワードが必要		バイト・ポート外部データワードが必要
	SIZ1	SIZ0	A1	A0	D31:D24	D23:D16	D15:D8	D7:D0	D31:D24	D23:D16	D31:D24
バイト	0	1	0	0	OP3	N	N1	N2	OP3	N	OP3
	0	1	0	1	PR	OP3	N	N1	PR	OP3	OP3
	0	1	1	0	PR1	PR	OP3	N	OP3	N	OP3
	0	1	1	1	PR2	PR1	PR	OP3	PR	OP3	OP3
ワード	1	0	0	0	OP2	OP3	N	N1	OP2	OP3	OP2
	1	0	0	1	PR	OP2	OP3	N	PR	OP2	OP2
	1	0	1	0	PR1	PR	OP2	OP3	OP2	OP3	OP2
	1	0	1	1	PR2	PR1	PR	OP2	PR	OP2	OP2
3バイト	1	1	0	0	OP1	OP2	OP3	N	OP1	OP2	OP1
	1	1	0	1	PR	OP1	OP2	OP3	PR	OP1	OP1
	1	1	1	0	PR1	PR	OP1	OP2	OP1	OP2	OP1
	1	1	1	1	PR2	PR1	PR	OP1	PR	OP1	OP1
ロング・ワード	0	0	0	0	OP0	OP1	OP2	OP3	OP0	OP1	OP0
	0	0	0	1	PR	OP0	OP1	OP2	PR	OP0	OP0
	0	0	1	0	PR1	PR	OP0	OP1	OP0	OP1	OP0
	0	0	1	1	PR2	PR1	PR	OP0	PR	OP0	OP0

注：Nn(次のn)およびPRn(前のn)となっているバイトは、キャッシュ可能なリード・サイクルでのみ有効となっている必要があります。これらは、キャッシュ不可能なリード・サイクルではdon't careとして解釈されます。

表7-5に、SIZ1、SIZ0、A1、およびA0の組合せと、対応するライト・サイクルにおけるMC68030の内部マルチプレクサから外部データ・バスへのデータ転送パターンをリストします。

図7-5に、ロング・ワード・オペランドをワード・ポートに転送する様子を示します。最初のバス・サイクルで、MC68030は4つのオペランド・バイトを外部バスに置きます。この例では、アドレスがロング・ワード境界に揃っていますので、マルチプレクサは、表7-5でSIZ0__SIZ1__A0__A1

表7-5　MC68030の内部から外部データ・バスへのマルチプレクサ——ライト・サイクル

転送サイズ	サイズ		アドレス		外部データ・バス接続			
	SIZ1	SIZ0	A1	A0	D31 : D24	D23 : D16	D15 : D8	D7 : D0
バイト	0	1	X	X	OP3	OP3	OP3	OP3
ワード	1	0	X	0	OP2	OP3	OP2	OP3
	1	0	X	1	OP2	OP2	OP3	OP2
3バイト	1	1	0	0	OP1	OP2	OP3	OP0*
	1	1	0	1	OP1	OP1	OP2	OP3
	1	1	1	0	OP1	OP2	OP1	OP2
	1	1	1	1	OP1	OP1	OP2*	OP1
ロング・ワード	0	0	0	0	OP0	OP1	OP2	OP3
	0	0	0	1	OP0	OP0	OP1	OP2
	0	0	1	0	OP0	OP1	OP0	OP1
	0	0	1	1	OP0	OP0	OP1*	OP0

*現在のインプリメンテーションでは、このバイトは出力されますが使用されません。
x = Don't Care
注：外部データ・バスの欄のOPテーブルは、データ・バスのその部分でリード/ライトされるオペランドの特定バイトを示しています。

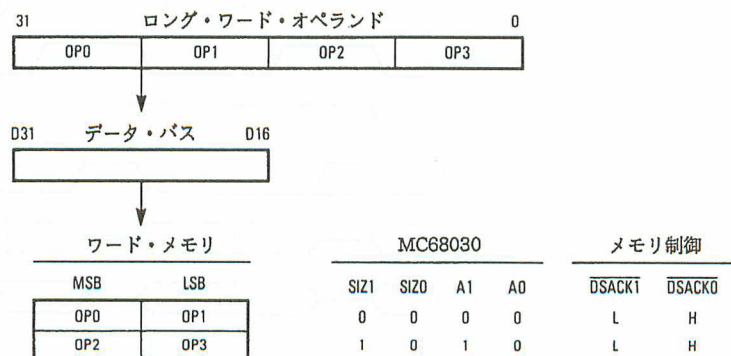



図7-5　ワード・ポートへのロング・ワード転送の例

＝0000に対応するエントリのパターンに従います。このポートはデータ・バスのビットD16－D31上のデータをラッチして、$\overline{\text{DSACK1}}$をアサートし($\overline{\text{DSACK0}}$はネゲートされたまま)、プロセッサがこのバス・サイクルを終了します。ついで、SIZ0__SIZ1__A0__A1＝1010として新しいバス・サイクルを開始し、残りの16ビットを転送します。サイズ信号は転送するワードが残っていることを示しています。A0とA1はワードがベース・アドレスからのオフセット2に対応することを示します。マルチプレクサは、このサイズおよびアドレス信号構成に対応するパターンに従い、ロング・ワードの最下位2バイトをバスのワード部分(D16～D31)に置きます。バス・サイクルは残りのバイトをワード・サイズのポートに転送します。図7-6にこの操作のバス転送信号のタイミングを示します。

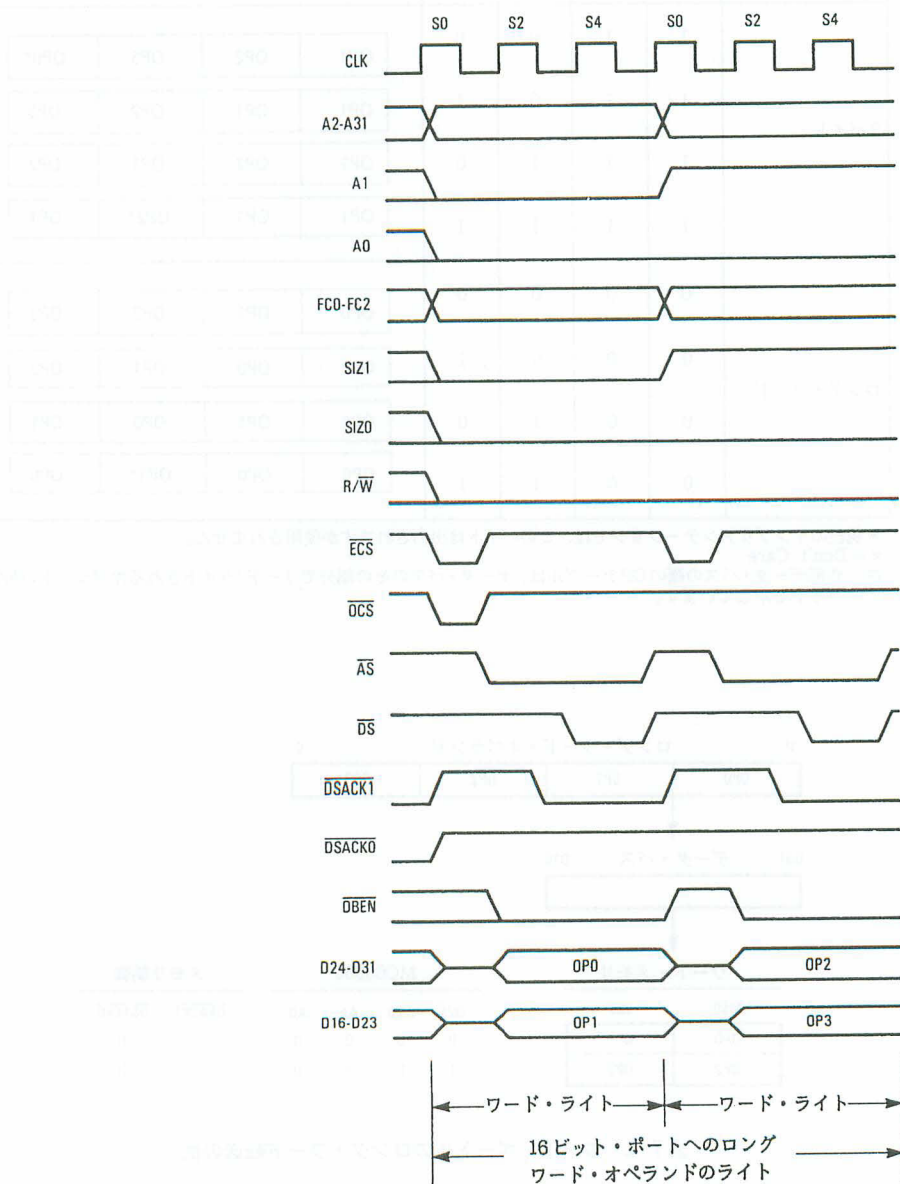



図7-6　ロング・ワード・オペランドのライト・タイミング(16ビット・データ・ポート)

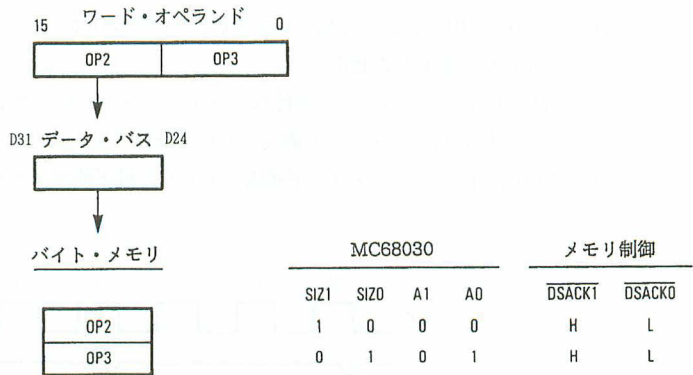



図7-7　バイト・ポートへのワード転送

図7-7に8ビット・バス・ポートへのワード転送の様子を示します。前出の例と同様に、この例も2バス・サイクルを要します。各バス・サイクルで1バイトを転送します。第1サイクルのサイズ信号は2バイトを指定し、第2サイクルは1バイトを指定します。図7-8に関連するバス転送信号のタイミングを示します。

7.2.2　ミスアラインメントのバス転送

オペランドはどのバイト境界にあってもよいため、ミスアラインメントが発生する可能性があります。バイト・オペランドは、どのアドレスにでも正しく整列しますが、ワード・オペランドは奇数アドレスではミスアラインメントになります。ロング・ワードは、4で割り切れないアドレスではミスアラインメントになります。MC68000、MC68008、およびMC68010では、奇数のワード境界へのロング・ワード転送は可能ですが、奇数のバイト・アドレスでワードまたはロング・ワード・オペランドを転送しようとすると例外が発生します。MC68030には、データ・オペランドのアラインメントに関する制限は(PC相対データ・アドレスも含めて)一切ありません。ただし、ミスアラインメントのロング・ワードまたはワード・オペランドのために余分なバス・アクセスが必要なときには、性能が低下してしまいます。性能を最大限に高めるには、データ・アイテムはそれぞれの自然な境界に整列していなければなりません。命令ワードとその拡張ワードは、ワード境界になければなりません。奇数アドレスで命令ワードをプリフェッチしようとするとアドレス・エラー例外が発生します。

図7-9にワード構成メモリで、ロング・ワード・オペランドを奇数アドレスへ転送する様子を示します。これには3バス・サイクルが必要です。最初のサイクルでは、サイズ信号はロング・ワード転送を示し、アドレス・オフセット(A2：A0)は001です。ポート幅は16ビットなので、ロング・ワードの最初のバイトだけが転送されます。スレーブ・デバイスはそのバイトをラッチして、データ転送の応答をし、ポートが16ビット幅であることを示します。プロセッサが第2サイクルを開始すると、サイズ信号はアドレス・オフセット(A2：A0)が010で、3バイトの転送が残っていることを示します。このサイクル中に次の2バイトが転送されます。ついで、プロセッサは第3サイクルを開始し、サイズ信号は1バイトの転送が残っていることを示します。このとき、アドレス・オフセット(A2：A0)は100となり、ポートは最後のバイトをラッチして、操作が完了します。図7-10に関連するバス転送信号のタイミングを示します。

図7-11に同じ操作をキャッシュ可能なデータ・リード・サイクルで実行した場合を示します。

図7-12と図7-13に、ワード構成メモリの奇数アドレスにワードを転送する例を示します。この例は図7-9と図7-10の例とよく似ていますが、オペランドがワード・サイズであり、転送には2バス・サイクルしか必要ありません。

図7-14に同じ操作をキャッシュ可能なデータ・リード・サイクルで実行した場合を示します。

図7-15と7-16にロング・ワード構成メモリの奇数アドレスにロング・ワードを転送する例を示します。この例では、ロング・ワード構成のメモリの最下位バイトからロング・ワード・アクセス

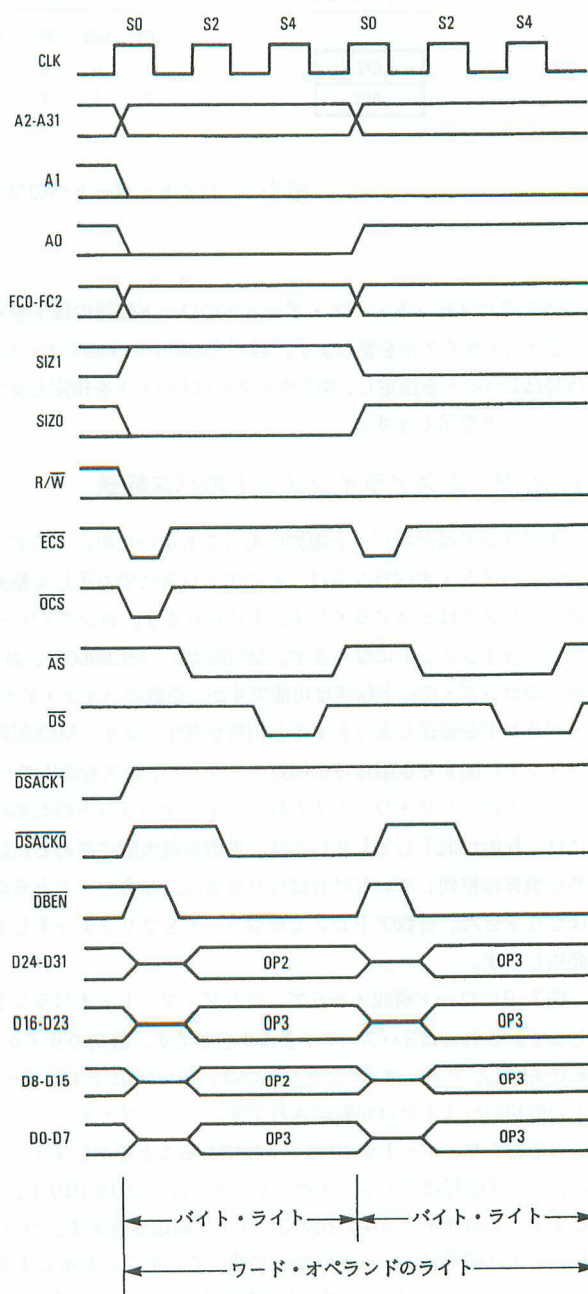



図7-8　ワード・オペランドのライト・タイミング(8ビット・データ・ポート)

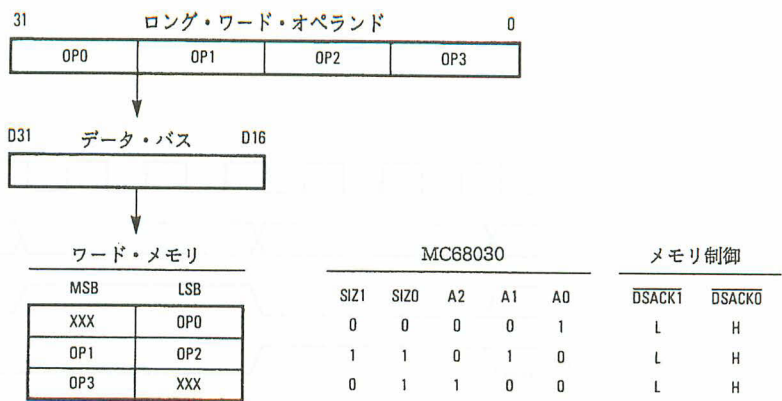



図7-9　ワード・ポートへのミスアラインド・ロング・ワード転送の例

を試みます。第1バス・サイクルでは、1バイトしか転送できません。第2バス・サイクルはロング・ワード境界への3バイトのアクセスとなります。メモリはロング・ワード構成なので、これ以上のバス・サイクルは不要です。

図7-17に同じ操作をキャッシュ可能なデータ・リード・サイクルで実行した場合を示します。

7.2.3　ダイナミック・バス・サイジングおよびオペランドのミスアラインメントの影響

オペランド・サイズ、オペランドのアラインメント、およびポート・サイズの組合せにより、特定のメモリ・アクセスを実行するのに必要なバス・サイクル数が決まります。表7-6に各種サイズのオペランドをサイズの違うポートを通じて転送するのに必要なバス・サイクル数を、ライト・サイクルとキャッシュ不可能なリード・サイクルに対するすべてのアラインメント状態について示します。

この表からわかるように、バス・サイクルのスループットはポート・サイズとアラインメントに大きく影響されます。MC68030のシステム設計者とプログラマは、この点に注意し、特に時間に厳しいアプリケーションではこの影響を十分考慮に入れておく必要があります。

表7-6はプロセッサがポート・サイズやアラインメントには関係なく、常にロング・ワード・アドレス(A2：A0＝000)からロング・ワードを読み出して、命令をプリフェッチすることを示しています。要求される命令が奇数ワード境界から始まるときには、プロセッサは2番目が必要なワードであっても、可能な場合は32ビット全体をフェッチし、両方のワードを命令キャッシュに入れるよう試みます。命令のアクセスがキャッシュに入っていない場合でも、32ビット全体が内部キャッシュ

表7-6　メモリのアラインメントとポート・サイズのライト・バス・サイクルへの影響

	バス・サイクル数			
A1/A0	00	01	10	11
命令＊	1：2：4	N/A	N/A	N/A
バイト・オペランド	1：1：1	1：1：1	1：1：1	1：1：1
ワード・オペランド	1：1：2	1：2：2	1：1：2	2：2：2
ロング・ワード・オペランド	1：2：4	2：3：4	2：2：4	2：3：4

データ・ポートのサイズは32ビット：16ビット：8ビット
＊命令のプリフェッチは常にロング・ワード境界から2ワード

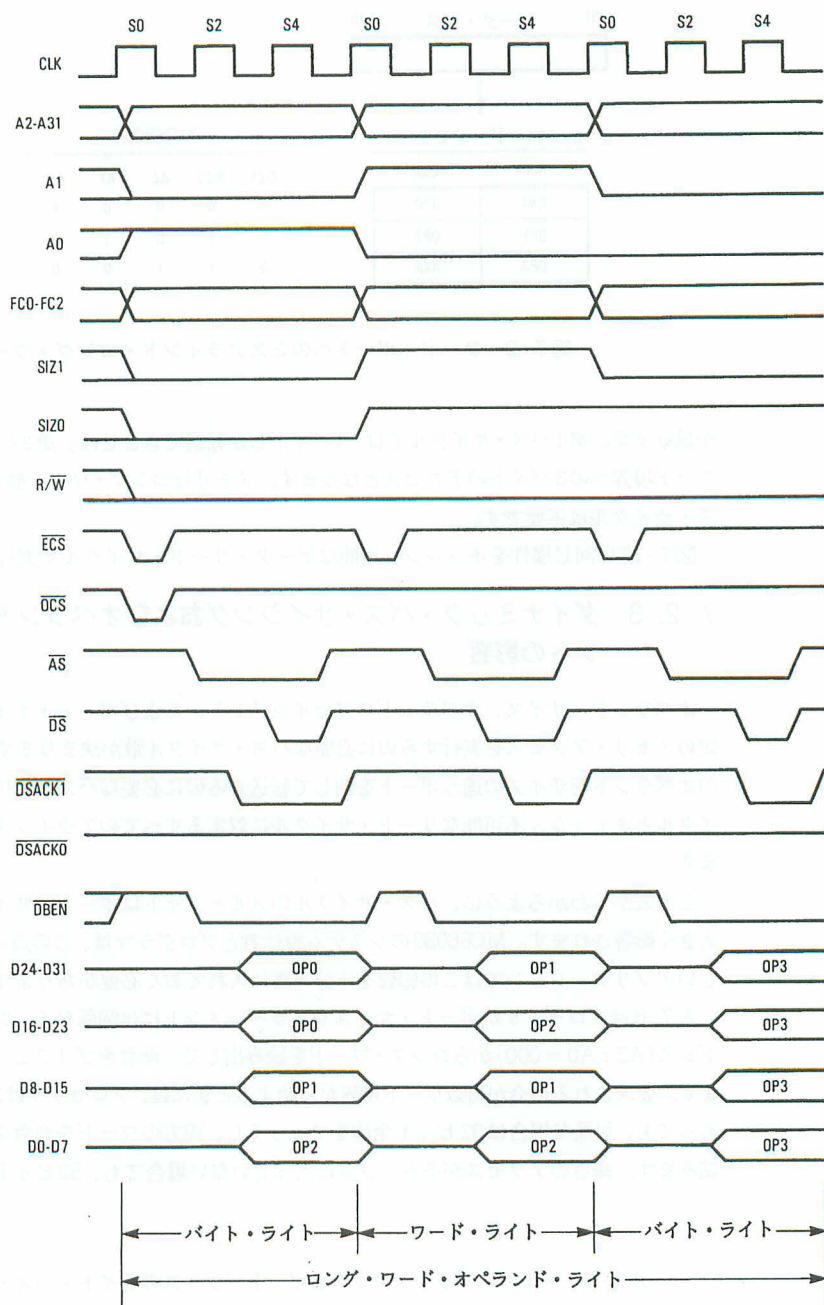



図7-10　ワード・ポートへのミスアラインド・ロング・ワード転送

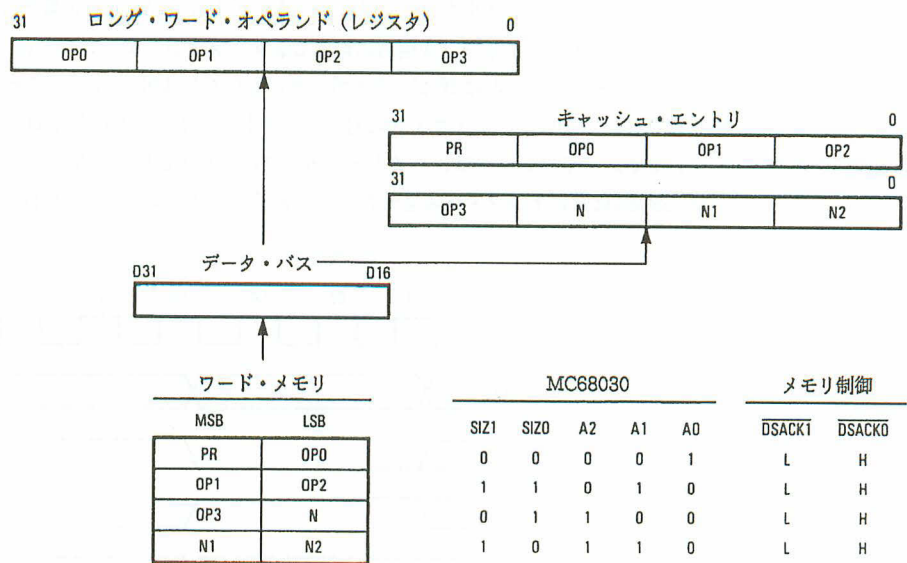



図7-11　ワード・ポートからのミスアラインド・キャッシュ可能ロング・ワード転送の例

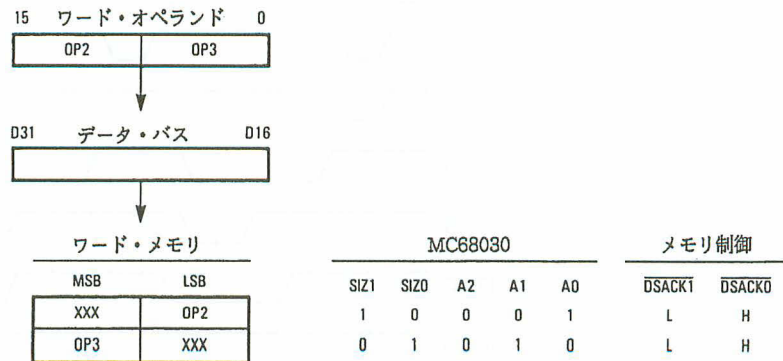



図7-12　ワード・ポートへのミスアラインド・ワード転送の例

保持レジスタにラッチされ、そこから2つの命令ワードを順次参照することができます。キャッシュ保持レジスタとパイプライン操作についての詳細は、「第11章　命令実行時間」を参照してください。

7. 2. 4　アドレス、サイズ、およびデータ・バスの関係

データ転送例をとりあげ、MC68030がどのようにデータ・バスの正しいバイト・セクションにデータを出力し、あるいはそこからデータを受け取るかを示します。表7-7にサイズ信号とアドレス信号A0およびA1の組合せを示します。これらは、アドレス指定されたデバイスが必要とするサイクルの種類すなわち、キャッシュ不可能なリード・サイクルとすべてのライト・サイクルで、データ・バスの4つのセクションのそれぞれに対して、バイト・イネーブル信号を生成するのに使用されます。

表に示すとおり、ポート・サイズもそれらのイネーブル信号の生成に影響を与えます。右側の4つのカラムは、4バイトのイネーブル信号に対応します。文字B、WまたはLはポート・サイズを示し、Bは8ビット・ポート、Wは16ビット・ポート、そしてLは32ビット・ポートを示しています。B、WまたはLは、そのポート・サイズに対してバイト・イネーブル信号が真でなければならないことを示します。ハイフン(－)は、バイト・イネーブル信号が適用できないことを示します。

MC68030は常にデータ・バス全セクションをドライブしますが、これはライト・サイクルの初め

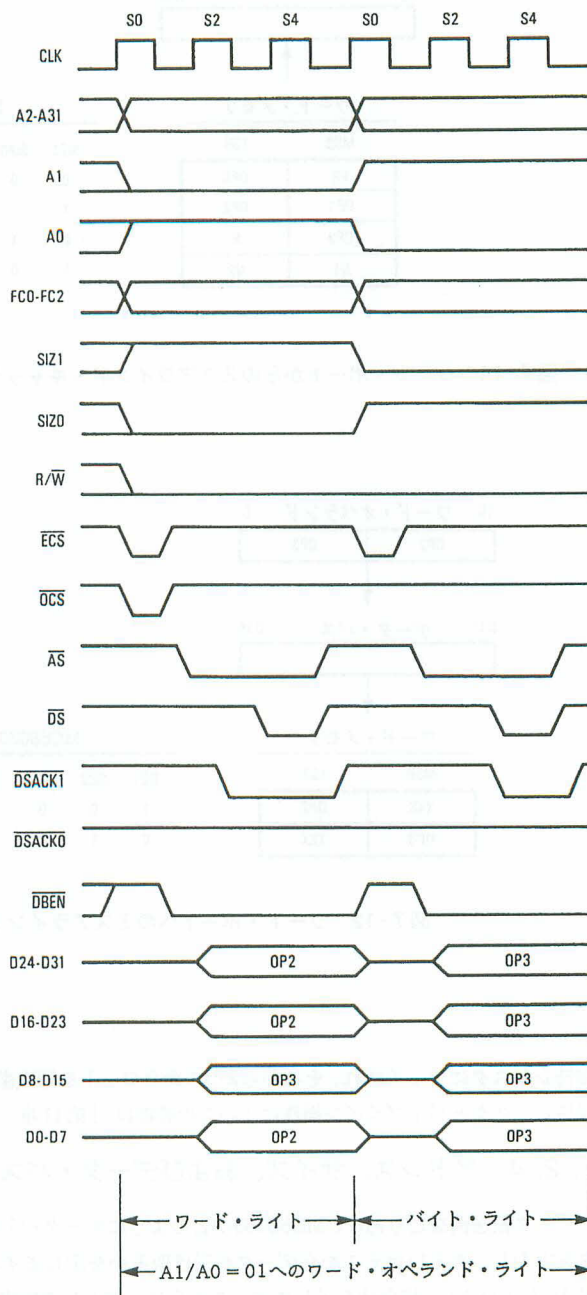



図7-13　ワード・ポートへのミスアラインド・ワード転送

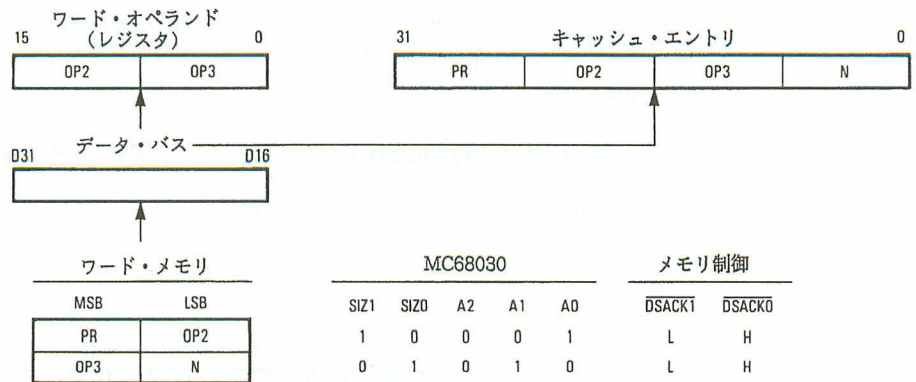



図7-14　ワード・バスからのミスアラインド・キャッシュ可能ワード転送の例

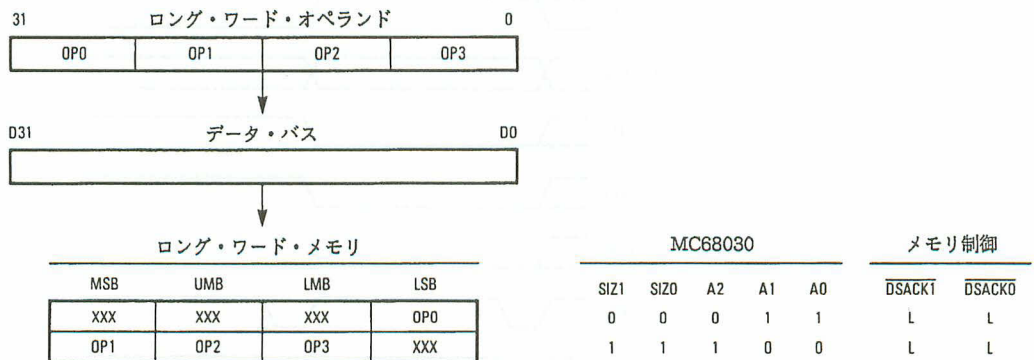



図7-15　ロング・ワード・ポートへのミスアラインド・ロング・ワード転送の例

ではバス・コントローラがポート・サイズを知らないためです。表のバイト・イネーブル信号は、内部でキャッシュされないリード操作とライト操作にだけ適用されます。キャッシュ可能なリード・サイクルでは、データがキャッシュされる間に、アドレス指定されたポートがそれが存在するバスの全セクションをドライブしなければなりません。

表を見ると、MC68030はオペランドがミスアラインメントになっていなかったり、バイト数がポート幅より大きくなければ、指定されたアドレスとの間で指定されたバイト数を転送することがわかります。このような場合、デバイスはそのポートに対して可能な最大のバイト数を転送します。

たとえば、サイズが4バイトで、アドレス・オフセット(A1：A0)が01の場合、32ビットのスレーブしか現在のバス・サイクルを受け取ることができません。16ビットまたは8ビットのスレーブは、1バイトしか受け取ることができません。この表はすべてのポート・サイズに対してバイト・イネーブルを定義します。バイト・データ・ストローブは、イネーブル信号とデータ・ストローブ信号を組み合わせて得られます。8ビット・ポートにあるデバイスは、各転送での有効バイトは1バイトしかないため、それ自身がデータ・ストローブを使用することができます。これらのイネーブルまたはストローブ信号は、ライト・サイクルまたはキャッシュ不可能なリード・サイクルに必要なバイトだけを選択します。他のバイトは選択されず、I/Oなどの微妙な領域で不当なアクセスが行なわれないようにしています。

図7-18に16ビットおよび32ビット・ポートに対して、サイズおよびアドレス・エンコーディング、そしてリード/ライト信号からバイト・データ・イネーブル信号を生成する方法を説明するための論理図を示します。

7. 2. 5 MC68030対MC68020のダイナミック・バス・サイジング

MC68030は非同期バス・サイクル($\overline{DSACKx}$で終了)に対して、MC68020のダイナミック・バス・

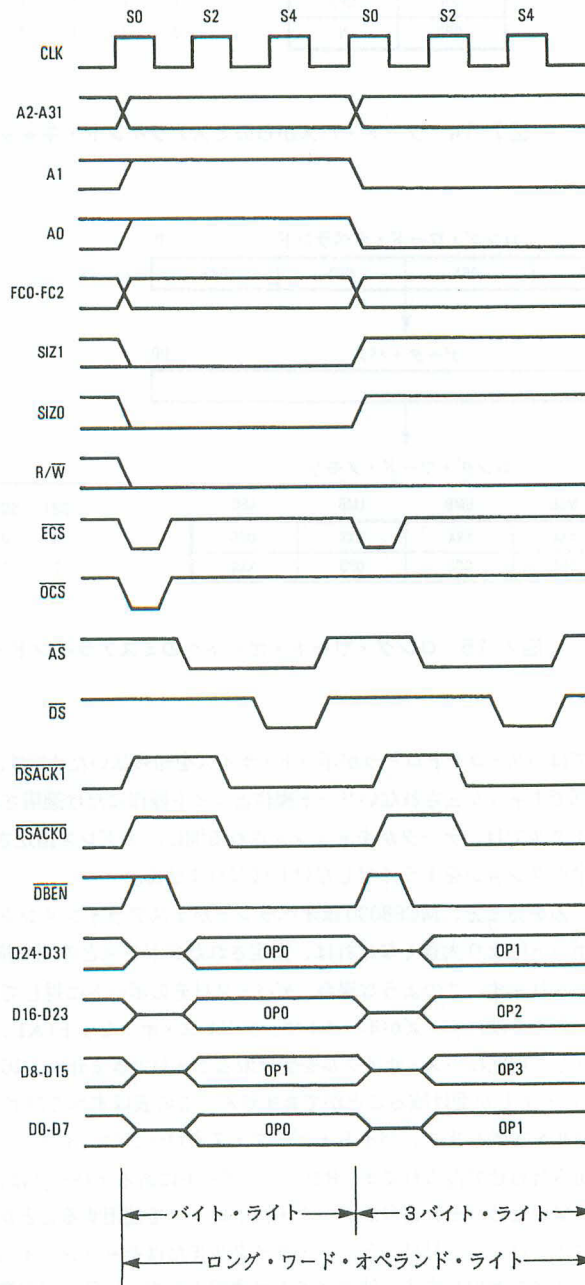



図7-16 ロング・ワード・ポートへのミスアラインド・ライト・サイクル

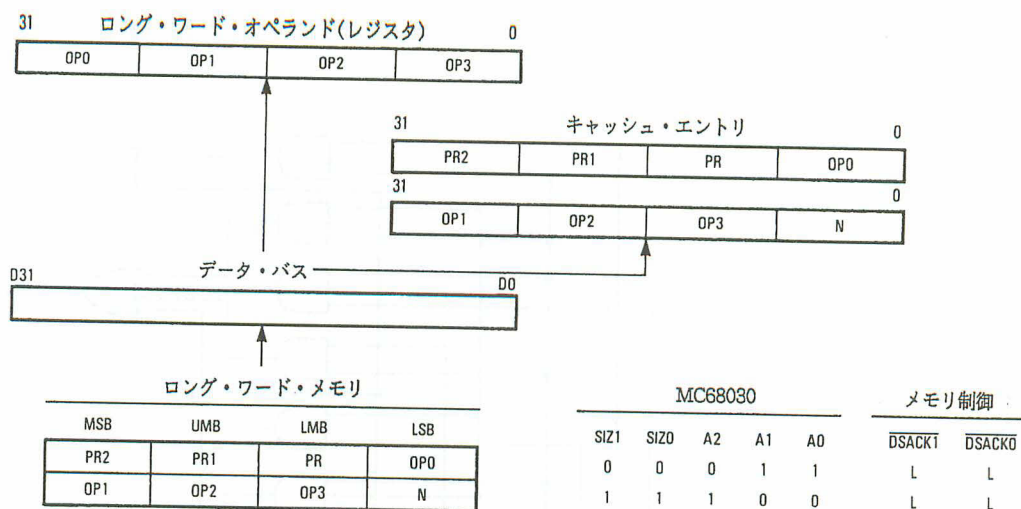



図7-17　ロング・ワード・バスからのミスアラインド・キャッシュ可能ロング・ワード転送の例

サイジングをサポートしています。ただし、これには次の2つの制約があります。その1つは、整列したロング・ワード境界内におけるキャッシュ可能なアクセスでは、各ロング・ワードの転送中に、ポート・サイズが一定していなければならないことです。たとえば、バイト・ポートがアドレス＄00にあるときは、アドレス＄01、＄02、および＄03もバイト・ポートに対応していなければなりません。もう1つの制約は、キャッシュ可能なアクセスに対しては、サイズ信号で示す転送サイズおよびA0とA1で示すアドレス・オフセットに関係なく、ポートはポート・サイズとして通知するデータ幅を供給しなければなりません。それ以外、MC68030とMC68020のダイナミック・バス・サイジングは同じです。

表7-7　バイト、ワード、およびロング・ワード・ポートに対するデータ・バス・ライト・イネーブル信号

転送サイズ	SIZ1	SIZ0	A1	A0	データ・バスのアクティブ部分 バイト(B) －ワード(W) －ロング・ワード(L) ポート			
					D31 : D24	D23 : D16	D15 : D8	D7 : D0
バイト	0	1	0	0	BWL	－	－	－
	0	1	0	1	B	WL	－	－
	0	1	1	0	BW	－	L	－
	0	1	1	1	B	W	－	L
ワード	1	0	0	0	BWL	WL	－	－
	1	0	0	1	B	WL	L	－
	1	0	1	0	BW	W	L	L
	1	0	1	1	B	W	－	L
3バイト	1	1	0	0	BWL	WL	L	－
	1	1	0	1	B	WL	L	L
	1	1	1	0	BW	W	L	L
	1	1	1	1	B	W	－	L
ロング・ワード	0	0	0	0	BWL	WL	L	L
	0	0	0	1	B	WL	L	L
	0	0	1	0	BW	W	L	L
	0	0	1	1	B	W	－	L

注：これらのセレクト・ラインはアドレス・デコード・ラインと組み合わせることができます。
あるいは、これらをすべて同じプログラム・アレイ・ロジック・ユニット内で生成することもできます。

図17-18　16および32ビット・ポートに対するバイト選択信号を発生する回路

7. 2. 6　キャッシュの充てん

「第6章　オンチップ・キャッシュ・メモリ」で説明するオンチップ・データ・キャッシュと命令キャッシュは、それぞれ4ロング・ワード・エントリの16ラインで構成されています。各ラインについて、タグが論理アドレスの最上位ビット、FC2(命令キャッシュ)またはFC0-FC2(データ・キャッシュ)、およびそのラインの各エントリに対する有効ビットをもっています。エントリ充てん操作は、メモリからアクセスされるロング・ワード全体をキャッシュ・エントリにロードします。この種の充てん操作は、ラインの1つのエントリが有効ではなく、アクセスがキャッシュ可能なときに実行されます。現在のサイクルでタグ・ミスが発生するか、キャッシュ・ラインの4つのエントリがすべて無効のとき(キャッシュがイネーブル、そしてそのキャッシュのバースト充てんがイネーブルの場合)、バースト充てん操作が要求されます。バースト充てん操作は、ラインにある4つのエントリをすべて充てんするよう試みます。バースト充てんをサポートするために、スレーブ・デバイスは32ビットのポートをもち、バースト・モード機能を備えていなければなりません。つまり、バースト要求に対し、キャッシュ・バースト・アクノリッジ($\overline{\mathrm{CBACK}}$)信号で認識応答を行なうことが必要です。また、バースト・アクセスを同期ターミネーション信号($\overline{\mathrm{STERM}}$)で終了させ、各転送ごとにロング・ワードをデータ・バス上に置かなければなりません。デバイスは、各ロング・ワード転送ごとに$\overline{\mathrm{STERM}}$をアサートしながら、キャッシュ・ラインがいっぱいになるまで、連続してロング・ワードを供給することができます。エントリ充てんおよびバースト・モード充てんによるキャッシュの充てんについての詳細は、「6. 1. 3　キャッシュの充てん」および「7. 3. 4　同期リード・サイクル」、「7. 3. 5　同期ライト・サイクル」、そして「7. 3. 7　バースト操作サイクル」を参照してください。これらの項では、要求されるバス・サイクルを詳細に説明しています。

7. 2. 7　キャッシュの相互作用

オンチップ命令キャッシュおよびデータ・キャッシュの構成と要求条件が、$\overline{\mathrm{DSACKx}}$および$\overline{\mathrm{STERM}}$信号の解釈に影響を与えることがあります。MC68030はキャッシュ可能なすべてのデータ・オペランドと命令をオンチップ・キャッシュにロードしようとするため、キャッシングがイネーブルされているときには、バスの動作が異なる場合があります。特に、通常どおり終了するキャッシュ可能なリード・サイクルでは、下位アドレス信号(A0およびA1)とサイズ信号は出力されません。

スレーブ・デバイスは、要求されたオペランド・サイズには関係なく、ポート・サイズで許容されるだけ、できるかぎり多くの整列したデータを供給しなければなりません。つまり、8ビット・ポートはバイト、16ビット・ポートはワード、そして32ビット・ポートはロング・ワード全体を供給しなければならないということです。このデータはキャッシュにロードされます。32ビット・ポートに対しては、スレーブ・デバイスはA0とA1を無視して、データ・バスのロング・ワード境界から始まるロング・ワードを供給します。16ビット・ポートに対しては、デバイスはA0を無視して、データ・バスのD16-D31の下位ワード境界から始まるワード全体を供給します。バイト・ポートに対しては、デバイスはD24-D31にアドレス指定されたバイトを供給します。

アドレス指定されたデバイスがポート・サイズ分のデータを供給できなかったり、データをキャッシュしてはならない場合、デバイスはリード・サイクルを終了したときに、キャッシュ・インヒビット・イン($\overline{\mathrm{CIIN}}$)をアサートしなければなりません。バス・サイクルが異常終了した場合、MC68030はデータをキャッシュしません。ポート・サイズの作用、ミスアラインメント、およびキャッシュの充てんについての詳細は、「6. 1. 3　キャッシュの充てん」を参照してください。

キャッシュはアドレス・ストローブ信号($\overline{\mathrm{AS}}$)のアサートやリード・サイクルの操作にも影響を与えることがあります。マイクロシーケンサが命令またはデータ・アイテムを要求すると、プロセッサが該当するキャッシュのサーチを始めます。このとき、命令キャッシュまたはデータ・キャッシ

ュに要求したアイテムが存在しなければ、バス・コントローラは外部バス・サイクルを開始することもできます。別のリードまたはライト・サイクルでバスが占有されていない場合、バス・コントローラは$\overline{\mathrm{ECS}}$信号(および場合によっては、$\overline{\mathrm{OCS}}$信号)をアサートします。内部キャッシュ・ヒットが起こると外部サイクルがアボートし、$\overline{\mathrm{AS}}$はアサートされません。これにより、複数の連続クロック・サイクルで、$\overline{\mathrm{ECS}}$をアサートすることができます。$\overline{\mathrm{ECS}}$をネゲートしてから、次に$\overline{\mathrm{ECS}}$をアサートするまでの最小時間が規定されています(「第13章　電気的特性」を参照のこと)。

命令のプリフェッチは1クロックおきに発生することがありますので、命令キャッシュのヒットによるアボート・サイクルの後、バス・コントローラが次のクロックで$\overline{\mathrm{ECS}}$をアサートした場合、この第2サイクルはデータ・フェッチのためのものになります。しかし、データ・キャッシュでヒットするデータ・アクセスも$\overline{\mathrm{ECS}}$のアサートとアボート・サイクルを発生する可能性があります。したがって、命令アクセスとデータ・アクセスは並行して行なわれるため、両方のキャッシュでヒットがあり、バス・コントローラが解放されている場合は、外部バスで$\overline{\mathrm{ECS}}$が何回か連続してアサートされます。なお、バス・コントローラが別のサイクルを実行中の場合、キャッシュ・ヒットのためにアボートされたサイクルは、外部からは見えません。また、$\overline{\mathrm{OCS}}$はオペランド転送の最初の外部サイクルでアサートされます。したがって、オペランドの最初の部分がキャッシュ・ヒットを起こし(ただし、バス・コントローラが外部サイクルを開始してそれをアボートしない)、2番目がキャッシュ・ミスを起こすミスアラインメントのデータ転送の場合、オペランドの2番目の部分で$\overline{\mathrm{OCS}}$がアサートされます。

7. 2. 8　非同期動作

MC68030のバスは非同期方式で使用することができます。この場合、バスに接続されている外部デバイスはMC68030のクロックとは異なる周波数のクロックで動作します。非同期動作は、ハンドシェーク・ライン($\overline{\mathrm{AS}}$、$\overline{\mathrm{DS}}$、$\overline{\mathrm{DSACK1}}$、$\overline{\mathrm{DSACK0}}$、$\overline{\mathrm{BERR}}$、および$\overline{\mathrm{HALT}}$)だけを使用してデータ転送を制御します。この方法を用いて、$\overline{\mathrm{AS}}$信号でバス・サイクルの開始を知らせ、$\overline{\mathrm{DS}}$をライト・サイクルでの有効データの条件として使用します。サイズ出力と下位アドレス・ラインA1、A0をデコードすることにより、データ・バスのアクティブ部分を選択するストローブ信号を作ることができます。これにより、スレーブ・デバイス(メモリまたは周辺デバイス)は、リード・サイクルでは要求されたデータをデータ・バスの指定された部分に出力し、ライト・サイクルではデータをラッチし、ポート・サイズに対応する$\overline{\mathrm{DSACK1}}$/$\overline{\mathrm{DSACK0}}$の組合せをアサートしてサイクルを終了させることによりこれに応答します。

どのスレーブ・デバイスも応答しなかったり、アクセスが不当であった場合、外部制御回路が$\overline{\mathrm{BERR}}$をアサートしてそのバス・サイクルをアボートするか、$\overline{\mathrm{BERR}}$と$\overline{\mathrm{HALT}}$信号をアサートして再試行を行ないます。

$\overline{\mathrm{DSACKx}}$信号は、スレーブ・デバイスからのデータがリード・サイクルで有効になる時点より前にアサートすることができます。$\overline{\mathrm{DSACKx}}$をデータより先行させてよい時間は、パラメータ#31で規定されます。非同期システムで有効データがプロセッサにラッチされることを保証するためには、このパラメータ#31の値を満たさなければなりません(タイミング・パラメータについては、「第13章　電気的特性」を参照してください)。なお、$\overline{\mathrm{AS}}$がアサートされてから$\overline{\mathrm{DSACKx}}$がアサートされるまでの時間の最大値は規定されていません。プロセッサはサイクルが$\overline{\mathrm{DSACKx}}$で終了するときには、データを最小3クロック・サイクルで転送することができますが、$\overline{\mathrm{DSACKx}}$が認識されるまでは、1クロック周期単位でウエイト・サイクルを挿入します。

$\overline{\mathrm{BERR}}$または$\overline{\mathrm{HALT}}$(あるいはその両方)は、$\overline{\mathrm{DSACKx}}$信号がアサートされた後アサートすることができます。非同期システムでは、$\overline{\mathrm{DSACKx}}$がアサートされてからパラメータ#48で規定される時間内に、$\overline{\mathrm{BERR}}$または$\overline{\mathrm{HALT}}$をアサートしなければなりません。この最大遅延時間を守らなかった

場合は、プロセッサが誤動作することがあります。

非同期リード・サイクルでは、CIINの値がバス・サイクルのステート4の立上りエッジで内部的にラッチされます。非同期リード・サイクルのステートについての詳細は、「7. 3. 1 非同期リード・サイクル」を参照してください。

DSACKxまたはBERRで終了するバス・サイクルでは、CBACKのアサートは完全に無視されます。

7. 2. 9 DSACKxとの同期動作

DSACKx信号で終了するサイクルは“非同期”に分類され、STERMで終了するサイクルは“同期”に分類されていますが、DSACKx信号で終了するサイクルも、信号がクロック・エッジを基準にして解釈されたため、同期方式で動作することができます。

これらのサイクルを使用するデバイスが同期動作を行なうには、応答をMC68030のクロックに同期させなければなりません。これらは、データ転送およびサイズ・アクノリッジ信号(DSACKx)でバス・サイクルを終了するため、MC68030のダイナミック・バス・サイジング機能を利用することができます。加えて、これらのサイクルの最小サイクルも同じく3クロックです。

システム・クロックを使用してDSACKxを発生したり他の非同期入力を行なうシステムをサポートするために、非同期入力セットアップ時間(パラメータ#47A)、および非同期入力ホールド時間(パラメータ#47B)が規定されています。DSACKxなどの信号をアサートまたはネゲートするときにセットアップ時間とホールド時間が満たされていれば、プロセッサがシステム・クロックの特定の立下りエッジでその信号レベルを正しく認識することが保証されます。DSACKxのアサート状態がクロックの立下りエッジで認識された場合、データがセットアップ時間(パラメータ#27)を満たしていれば、クロックの次の立下りエッジで有効データがプロセッサの中にラッチされます(リード・サイクルのとき)。この場合、非同期動作のためのパラメータ#31は無視できます。これらのタイミング・パラメータについては、「第13章 電気的特性」を参照してください。なお、システムがS2の立下りエッジで要求されるウインドの間にDSACKxをアサートし、クロック・エッジでASがネゲートされるまでDSACKx(およびBERR/HALTあるいは、そのいずれか)を保持することによって適切なバス・プロトコルに従っている場合(該当する非同期入力ホールド時間はパラメータ#47Bで規定される)、ウエイト・ステートは挿入されません。バス・サイクルは、1サイクル3クロックのDSACKxで終了するバス・サイクルにおいて最大速度で実行されます。

DSACKxの後でBERRまたはBERRとHALTがアサートされる同期システムを正しく動作させるには、BERR(およびHALT)はDSACKxが認識される1クロック・サイクル後のクロックの立下りエッジに対するセットアップ時間(パラメータ#27A)を満たしていなければなりません。このセットアップ時間は重要であり、これが満たされていないと、MC68030が誤動作することがあります。

同期動作を行なっているときには、DS信号に対するデータのタイミング条件の代わりに、同期サイクルに対するデータ入力セットアップ時間およびホールド時間を使用することができます。

CIINの値は、DSACKxで終了する全サイクルで、バス・サイクル・ステート4の立上りエッジでラッチされます。

7. 2. 10 STERMとの同期動作

MC68030は同期ターミネーション信号(STERM)で終了する同期バス・サイクルをサポートします。これらのサイクルは32ビット・ポート専用ですが、DSACKxで終了するサイクルとよく似ています。主な違いは、STERMはDSACKxで終了するサイクルよりも前にアサート(およびデータ転送)できるため、プロセッサは2クロック周期の最小アクセス時間で転送を行なうことができます。ただし、STERMのアサートを遅くすれば、ウエイト・サイクルを挿入することも可能です。

データ転送およびサイズ・アクノリッジ信号DSACKxの代わりに、同期終了信号STERMを使用すれば、どのバス・サイクルでも同期式にすることができます。バス・サイクルは、次の場合に同期式になります。

1. そのサイクル中に、DSACKxもオートベクタ信号AVECもアサートされない。
2. ポート・サイズが32ビットである。
3. 同期入力セットアップ時間およびホールド時間条件(仕様＃60および＃61)が満たされる。

バースト動作モードでは、STERMを使用してその各サイクルを終了しなければなりません。バースト転送の最初のサイクルは、上記パラグラフで説明したように、同期サイクルでなければなりません。このサイクルの正確なタイミングは、STERMをアサートすることによって制御され、必要に応じてウエイト・サイクルを挿入することができます。ただし、最小サイクル時間は2クロックです。バースト操作が開始され、通常どおり終了できる場合は、第2、第3、そして第4サイクルで、クロックの立下りエッジにより連続してデータをラッチすることができます。この場合も、後続サイクルの正確なタイミングは、これらの各サイクルに対するSTERMのタイミングで制御され、必要に応じてウエイト・サイクルを挿入することもできます。

同期入力信号(STERM、CIIN、およびCBACK)は、ASがアサートされているときには、クロックの立上り時間に対し、規定されるセットアップ時間およびホールド時間の間安定していなければなりません。CBACKおよびCIINのアサートまたはネゲートは、同期サイクルにおいて、STERMがアサートされるクロックの立上りエッジで内部にラッチされます。

STERM信号はアドレス・バスとファンクション・コード値から生成することができ、AS信号を条件として使用する必要はありません。STERMがアサートされ、サイクルが進行中でない場合(サイクルが開始され、ECSがアサートされた後そのサイクルがアボートされた場合でも)、MC68030はSTERMを無視します。

同様に、CBACKはCBREQのアサートとは関係なくアサートすることができます。キャッシュ・バーストが要求されていない場合、CBACKのアサートは無視されます。

CIINのアサートは、該当するキャッシュがイネーブルになっていないとき、またはCIOUTがアサートされたときには無視されます。ライト・サイクル中または変換テーブル・サーチ中にも無視されます。

注：STERMとDSACKxが同じバス・サイクルでアサートされてはなりません。

7. 3　データ転送サイクル

プロセッサと他のデバイス間でのデータ転送には、次の信号が関係します。

- ●アドレス・バスA0～A31
- ●データ・バスD0～D31
- ●制御信号

アドレス・バスおよびデータ・バスは、両方とも並列の非多重化バスです。バス・マスタは制御信号を出してバス上のデータを転送し、非同期/同期バスはハンドシェイク・プロトコルを使用して、正しいデータ転送を保証します。すべてのバス・サイクルで、バス・マスタにはサイクルの始めと終わりの両方で出力されるすべての信号に対してスキューを回避する責任があります。さらに、バス・マスタはスレーブ・デバイスからのアクノリッジ信号およびデータ信号のスキューも回避しなければなりません。以下のパラグラフでは、リード、ライト、およびリード・モディファイ・ライト・サイクル操作を定義します。また、バースト・モード転送についても説明しています。

各バス・サイクルは一連のステートとして定義されます。これらのステートは、バス操作に適用されるもので、「第4章　処理状態」で述べるステートとは異なります。データ転送サイクルの説明

およびタイミング図で使用するクロック・サイクルは、クロック周波数には関係ありません。バス操作は外部バスの状態を基準にして説明しています。

7.3.1　非同期リード・サイクル

リード・サイクルでは、プロセッサはメモリ、コプロセッサ、または周辺デバイスからデータを受け取ります。命令でロング・ワード操作が指定された場合、MC68030は一度に4バイトを読み出そうとします。ワード操作では一度に2バイト、バイト操作では1バイトを読み出そうとします。操作によっては、プロセッサは3バイトの転送を要求します。プロセッサは各バイトを内部で正しく位置決めします。データ・バスのどの部分から各バイトを読み出すかは、オペランドのサイズ、アドレス信号A0およびA1、$\overline{CIIN}$および$\overline{CIOUT}$、内部キャッシュがイネーブルされているか否か、そしてポート・サイズによって決まります。ダイナミック・バス・サイジング、オペランドのミスアラインメント、そしてキャッシュの相互作用についての詳細は、「7.2.1　ダイナミック・バス・サイジング」、「7.2.2　ミスアラインメントのバス転送」、そして「7.2.6　キャッシュの充てん」をそれぞれ参照してください。

図7-19に非同期ロング・ワード・リード・サイクルのフローチャートを示します。図7-20はバイト・リード・サイクルのフローチャートです。以下の図では、クロックの周期を基準にして規定される、リード・サイクルの機能タイミングを示します。図7-21は32ビット・ポートからのバイトおよびワード・リード・サイクルに対応します。図7-22は8ビット・ポートからのロング・ワードのリード・サイクルです。図7-23もロング・ワードのリード・サイクルですが、これは16ビット・ポートからのものです。

ステート0

リード・サイクルはステート0(S0)から始まります。プロセッサは外部サイクル・スタート($\overline{ECS}$)を“L”にドライブして、外部サイクルの開始を知らせます。このサイクルがリード操作での最初の外部サイクルのときは、同時にオペランド・サイクル・スタート($\overline{OCS}$)も“L”にドライブします。S0の間、プロセッサはアドレス・バス(A0-A31)に有効なアドレスを置き、FC0-FC2に有効なファンクション・コードを置きます。ファンクション・コードはそのサイクルのアドレス空間を選択します。プロセッサはリード・サイクルの間、リード/ライト(R/$\overline{W}$)を“H”にドライブし、データ・バッファ・イネーブル($\overline{DBEN}$)を非アクティブにドライブして、データ・バッファをディセーブルします。サイズ信号SIZ0とSIZ1が有効となり、転送に必要なバイト数を示します。キャッシュ・インヒビット・アウト($\overline{CIOUT}$)も有効となり、アドレス変換ディスクリプタまたは該当するTTxレジスタのMMU CIビットの状態を示します。

ステート1

半クロック後のステート1(S1)では、プロセッサはアドレス・ストローブ($\overline{AS}$)をアサートし、アドレス・バス上のアドレスが有効であることを示します。S1では、プロセッサはデータ・ストローブ($\overline{DS}$)もアサートします。また、S1では$\overline{ECS}$(および$\overline{OCS}$がアサートされている場合は$\overline{OCS}$)信号がネゲートされます。

ステート2

ステート2(S2)では、プロセッサは$\overline{DBEN}$をアサートして外部データ・バッファをイネーブルにします。選択されたデバイスは、R/$\overline{W}$、SIZ0-SIZ1、A0-A1、$\overline{CIOUT}$、および$\overline{DS}$を使用してその情報をデータ・バスに置き、必要ならキャッシュ・インヒビット・イン($\overline{CIIN}$)をドライブします。サイズ信号とA0－A1によって、(D24-D31、D16-D23、D8-D15およびD0-D7)の任意またはすべてのバイトが選択されます。同時に、選択されたデバイスがデータ転送およびサイズ・アクノリッジ($\overline{DSACKx}$)信号をアサートします。

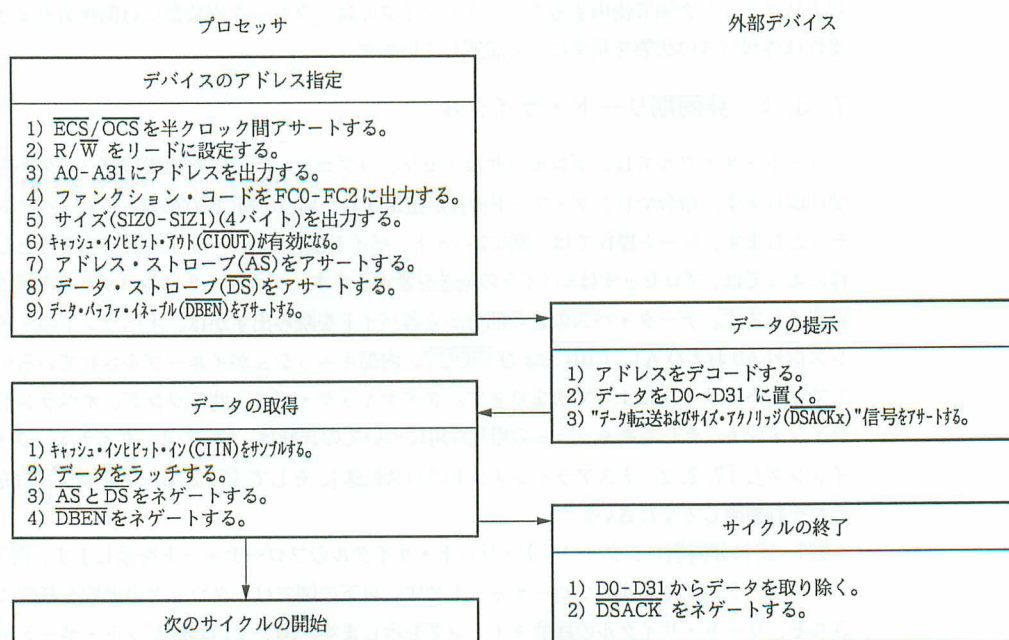



図7-19　非同期ロング・ワード・リード・サイクルのフローチャート

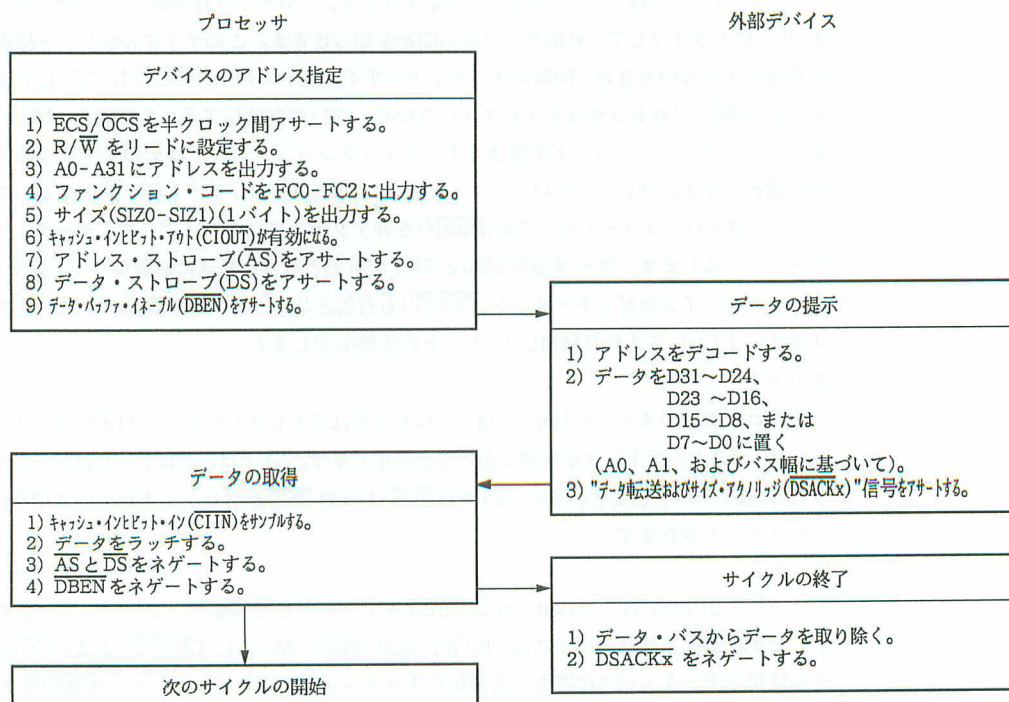



図7-20　非同期バイト・リード・サイクルのフローチャート

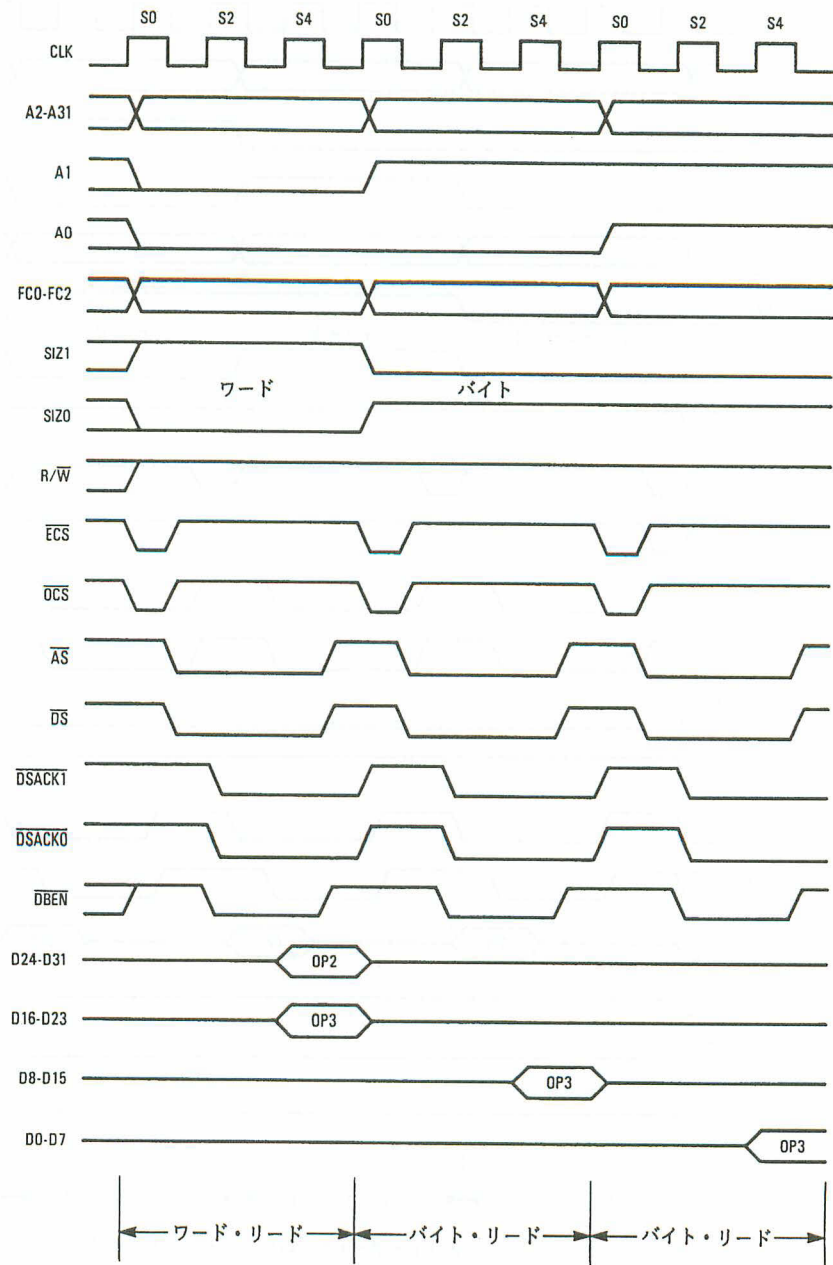



図7-21　非同期バイトおよびワード・リード・サイクル――32ビット・ポート

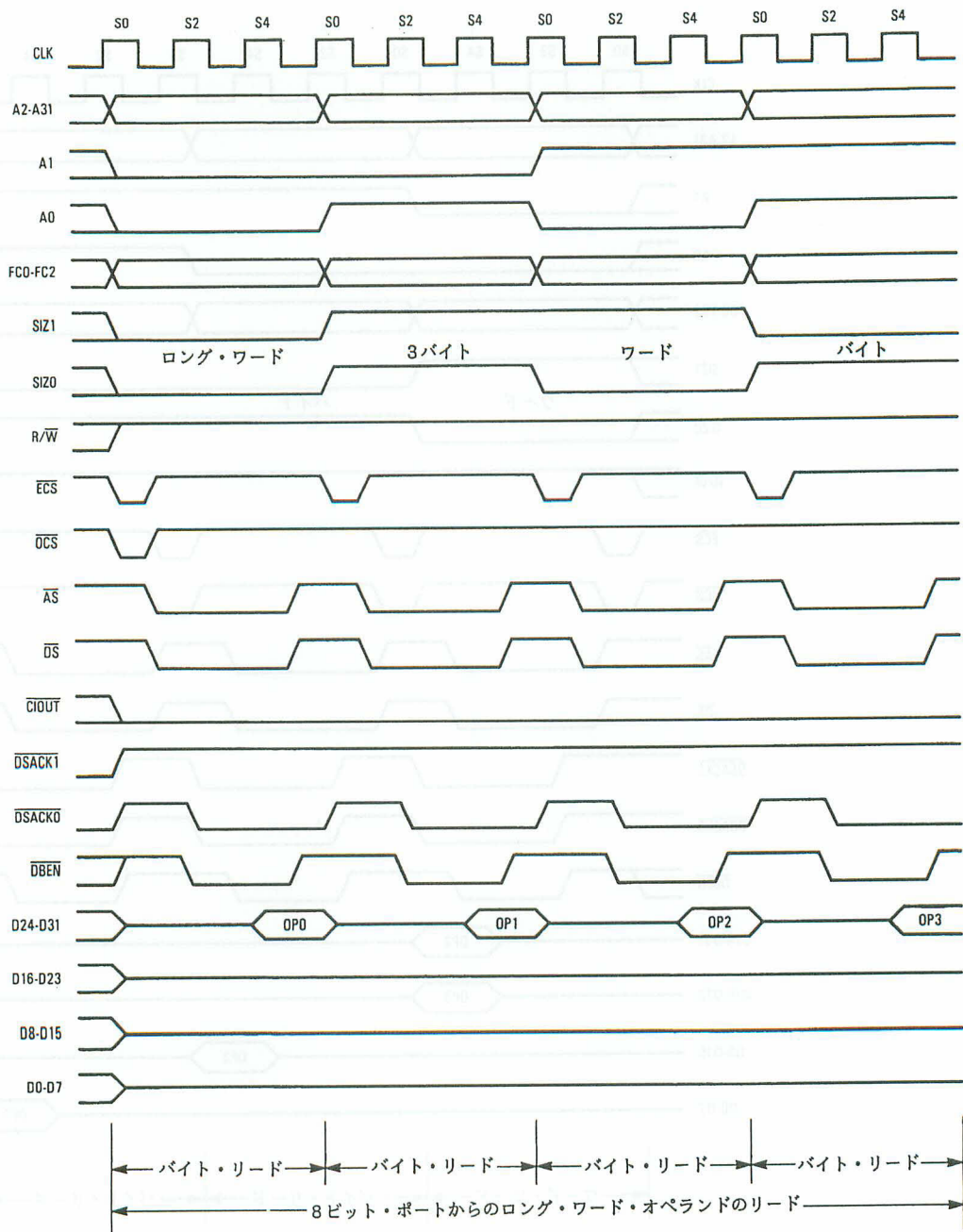



図7-22 ロング・ワード・リード——8ビット・ポートからのロング・ワード・オペランドのリード（$\overline{\text{CIOUT}}$アサート）

ステート3

S2の終わりで、少なくとも1つの$\overline{\mathrm{DSACKx}}$信号が認識されると(非同期の入力セットアップ時間条件件を満たすもの)、クロックの次の立下りエッジでデータがラッチされ、サイクルが終了します。ステート3(S3)の初めに$\overline{\mathrm{DSACKx}}$が認識されなかった場合、プロセッサはステート4および5に進まずに、ウエイト・ステートを挿入します。ウエイト・ステートを挿入するには、$\overline{\mathrm{DSACK0}}$と$\overline{\mathrm{DSACK1}}$の両方が、S2の終了付近で非同期入力のセットアップおよびホールド時間の間ネゲートされたままでなければなりません。ウエイト・ステートが追加されると、プロセッサは$\overline{\mathrm{DSACKx}}$信号の1つを認識するまで、クロックの立下りエッジで継続してサンプリングを行ないます。

ステート4

プロセッサはステート4(S4)の初めに$\overline{\mathrm{CIIN}}$をサンプリングします。$\overline{\mathrm{CIIN}}$は同期入力として定義されていますので、アサートされているときもネゲートのときも、$\overline{\mathrm{AS}}$がアサートされている間は、クロックの立上りおよび立下りエッジにおいて、所定の同期入力セットアップ時間およびホールド時間を満たしていなければなりません。S4の終わりで、プロセッサは入力データをラッチします。

ステート5

プロセッサはステート5(S5)で、$\overline{\mathrm{AS}}$、$\overline{\mathrm{DS}}$、および$\overline{\mathrm{DBEN}}$をネゲートします。S5の間はアドレスを有効に保持し、メモリ・システムに対してアドレス・ホールド時間を与えます。S5の期間中、R/$\overline{\mathrm{W}}$、SIZ1およびSIZ0、そしてFC0-FC2も有効になったままです。

外部デバイスは、$\overline{\mathrm{AS}}$または$\overline{\mathrm{DS}}$のネゲートを(どちらか最初に)検出するまで、データと$\overline{\mathrm{DSACKx}}$信号をアサートしたままにしておきます。$\overline{\mathrm{AS}}$または$\overline{\mathrm{DS}}$のネゲートを検出したあと、約1クロック期間以内にデータを取り除き、$\overline{\mathrm{DSACKx}}$信号をネゲートしなければなりません。この期間を超えてアサートしたままにしておくと、次のバス・サイクルですぐ検出されてしまうことがあります。

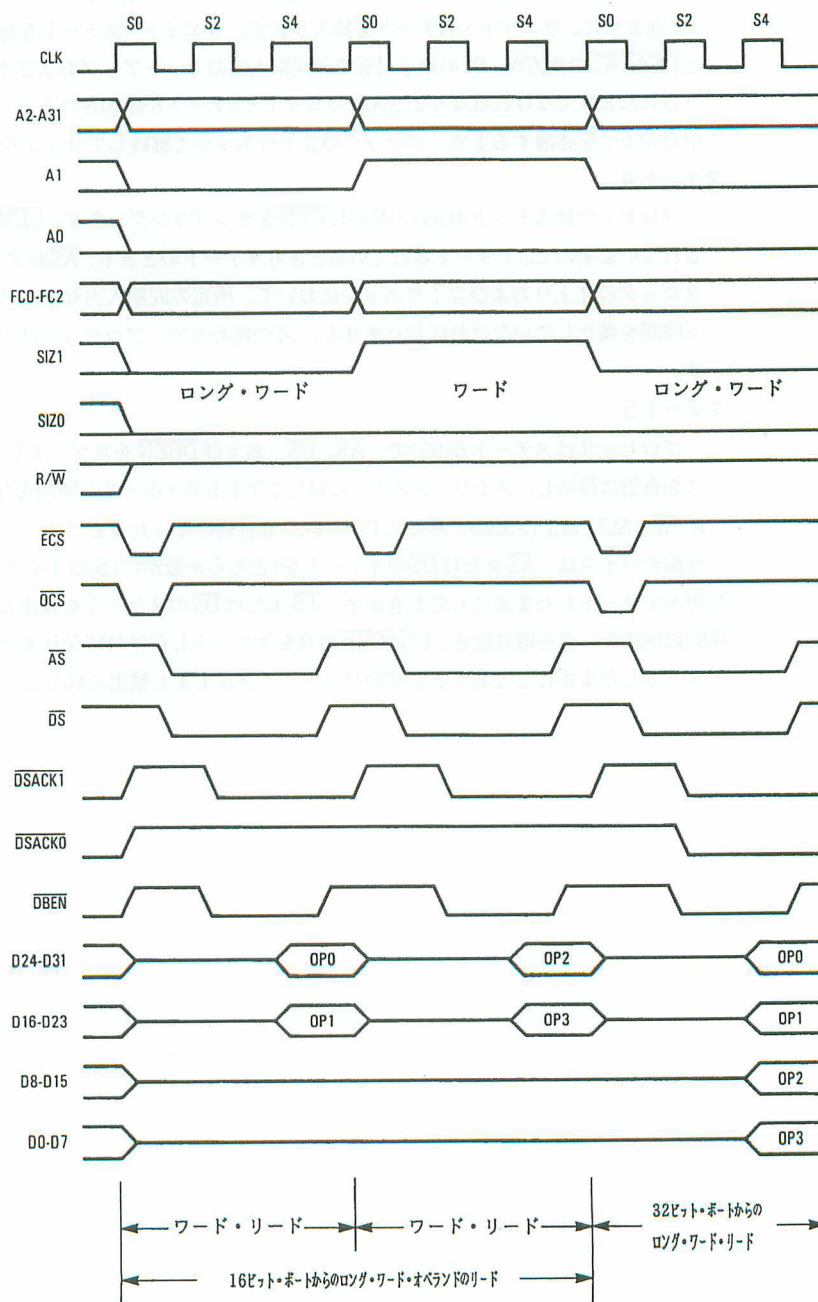



図7-23　ロング・ワード・リード——16ビット・ポートおよび32ビット・ポート

7.3.2　非同期ライト・サイクル

ライト・サイクルでは、プロセッサはデータをメモリまたは周辺デバイスに転送します。

図7-24にロング・ワード転送でのライト・サイクルのフローチャートを示します。以下の図では、クロックの周期を基準にして規定されるライト・サイクルの機能タイミングを示します。図7-25は32ビット・ポートに対する2つのライト・サイクル(2つのリード・サイクルに囲まれ、その間にはアイドル時間はない)を示します。図7-26に32ビット・ポートへのバイトおよびワード・ライト・サイクルを示します。図7-27は8ビット・ポートへのロング・ワードのライト・サイクルです。図7-28は16ビット・ポートへのロング・ワードのライト・サイクルです。

ステート0

ライト・サイクルはステート0(S0)から始まります。プロセッサは外部サイクル・スタート($\overline{ECS}$)を“L”にドライブして、外部サイクルの開始を知らせます。このサイクルがライト操作での最初の外部サイクルのときは、同時にオペランド・サイクル・スタート($\overline{OCS}$)も“L”にドライブします。S0の間、プロセッサはアドレス・バス(A0-A31)に有効なアドレスを置き、FC0-FC2に有効なファンクション・コードを置きます。ファンクション・コードでそのサイクルのアドレス空間を選択します。プロセッサはライト・サイクルの間、リード/ライト(R/$\overline{W}$)を“L”にドライブします。サイズ信号SIZ0とSIZ1が有効となり、転送するバイト数を示します。キャッシュ・インヒビット・アウト($\overline{CIOUT}$)も有効となり、アドレス変換ディスクリプタまたは該当するTTxレジスタのMMU　CIビットの状態を示します。

ステート1

半クロック後のステート1(S1)では、プロセッサはアドレス・ストローブ($\overline{AS}$)をアサートし、アドレス・バス上のアドレスが有効であることを示します。S1では、プロセッサはデータ・バッフ

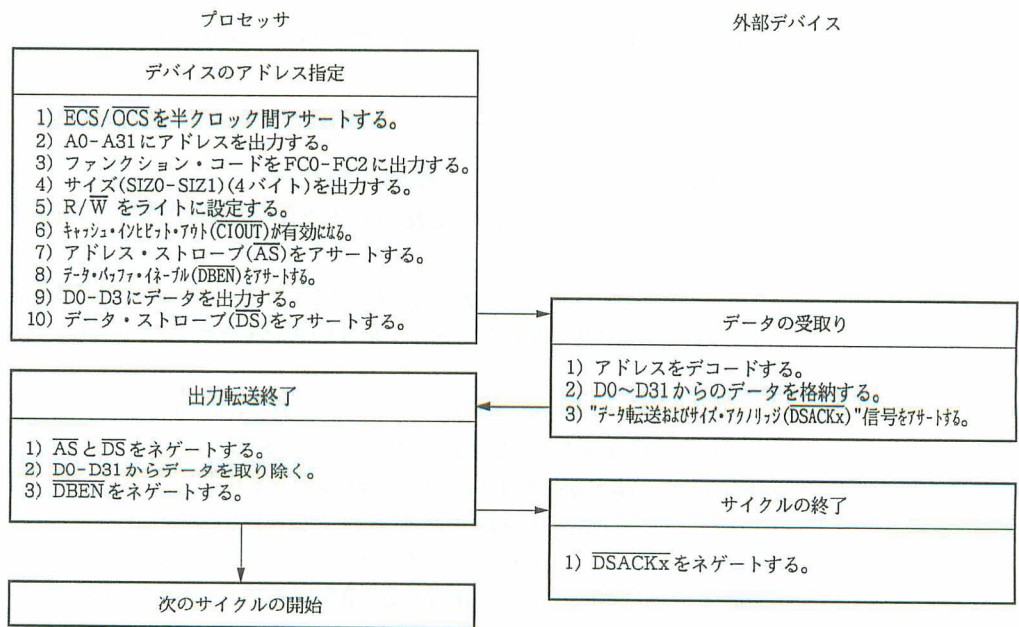



図7-24　非同期ライト・サイクルのフローチャート

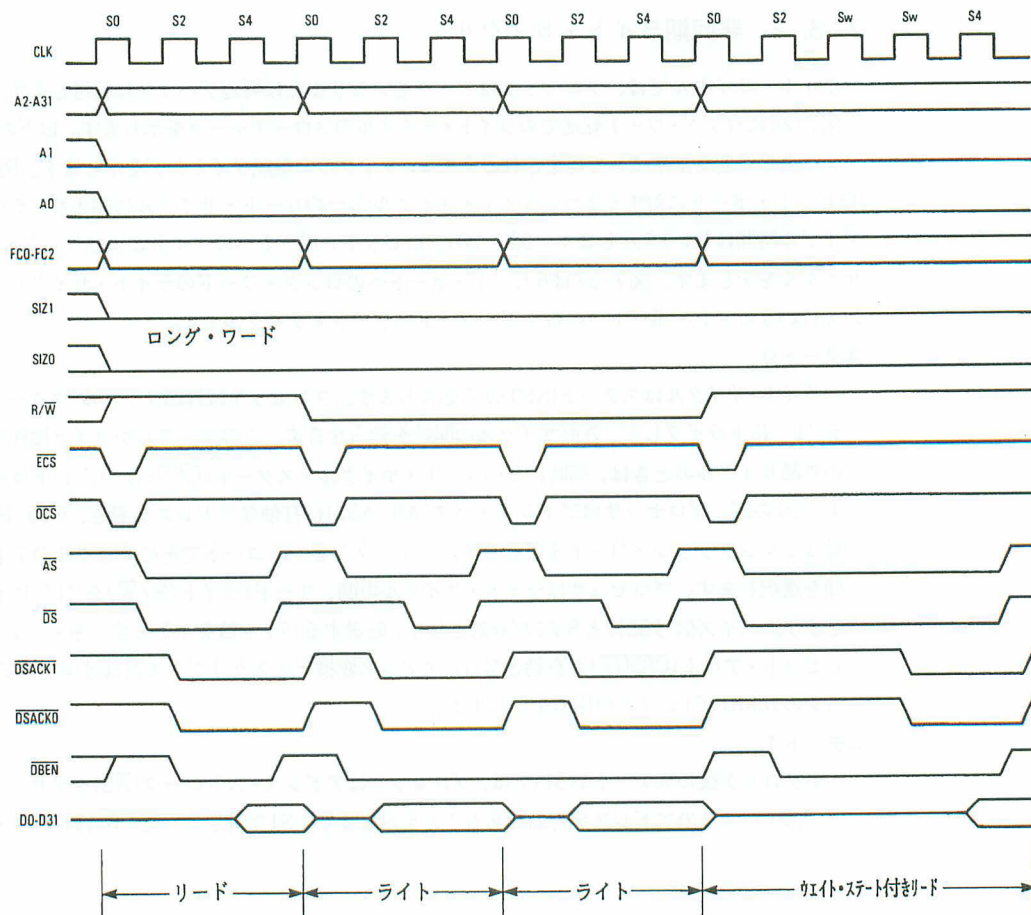



図7-25　非同期リード・ライト・リード・サイクル——32ビット・ポート

ァ・イネーブル($\overline{\text{DBEN}}$)もアサートし、これによって外部データ・バッファがイネーブルになります。また、S1では$\overline{\text{ECS}}$(およびアサートされている場合は$\overline{\text{OCS}}$)信号がネゲートされます。

ステート2

ステート2(S2)では、プロセッサは書き込むデータをデータ・バス(D0-D31)に置き、S2の終わりで、データ転送およびサイズ・アクノリッジ信号$\overline{\text{DSACKx}}$をサンプリングします。

ステート3

プロセッサはステート3(S3)でデータ・ストローブ($\overline{\text{DS}}$)をアサートします。この信号は、データ・バス上でデータが安定していることを示します。S2の終わりで、少なくとも1つの$\overline{\text{DSACKx}}$信号が認識されると(非同期の入力セットアップ条件を満たすもの)、1クロック後にこのサイクルが終了します。ステート3(S3)の初めに$\overline{\text{DSACKx}}$が認識されない場合、プロセッサはステート4および5に進まずに、ウエイト・ステートを挿入します。ウエイト・ステートを挿入するには、$\overline{\text{DSACK0}}$と$\overline{\text{DSACK1}}$の両方が、S2の終了付近で非同期入力のセットアップおよびホールド時間の間ネゲートされたままでなければなりません。ウエイト・ステートが追加されると、プロセッサは$\overline{\text{DSACKx}}$信号の1つを認識するまで、クロックの立下りエッジで継続してサンプリングを行ないます。選択されたデバイスは、R/$\overline{\text{W}}$、$\overline{\text{DS}}$、SIZ0-SIZ1、およびA0-A1を使用して、データ・

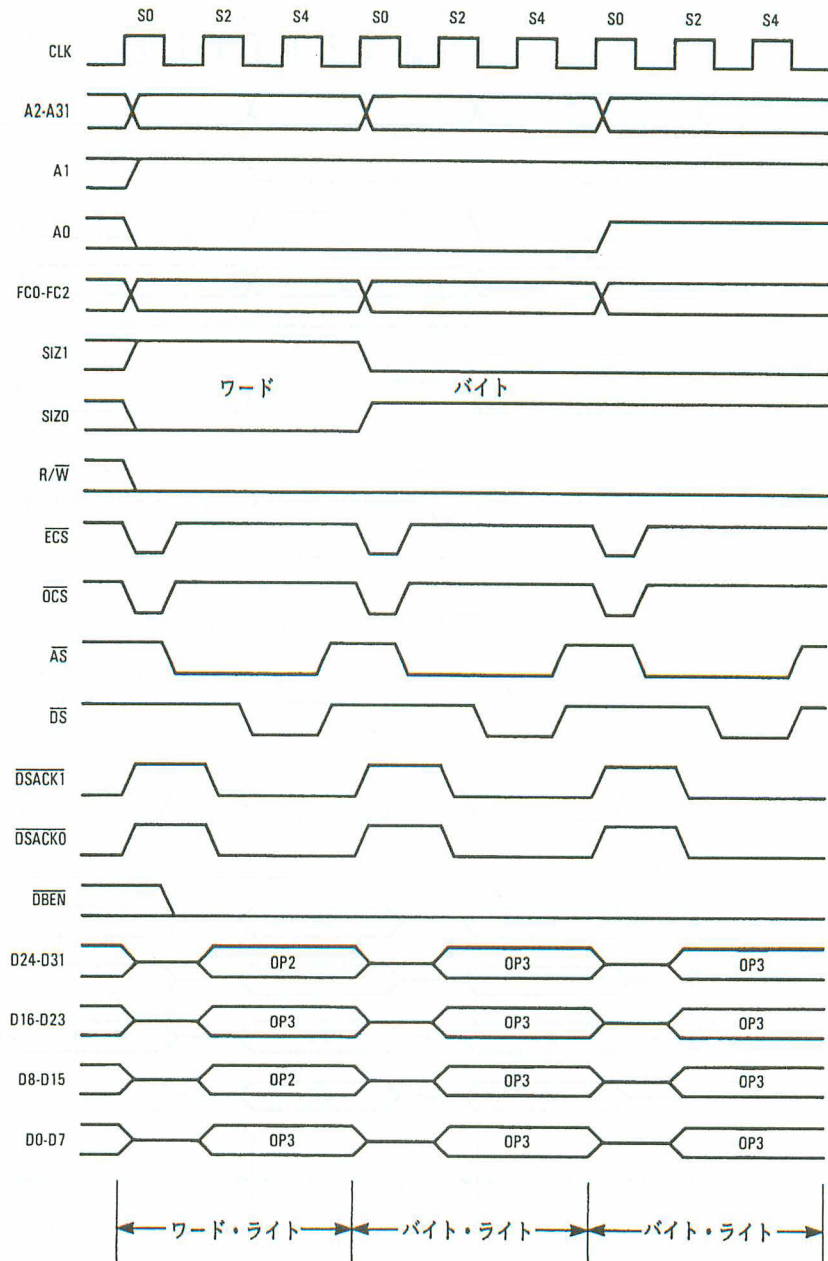



図7-26 非同期バイトおよびワード・ライト・サイクル――32ビット・ポート

バス(D24-D31、D16-D23、D8-D15、およびD0-D7)の該当するバイトから送られたデータをラッチします。サイズ信号、A0-A1によってデータ・バス上のバイトを選択します。まだ$\overline{\text{DSACKx}}$がアサートされていなければ、デバイスはそれをアサートして、正常にデータを格納したことを知らせます。

ステート4

プロセッサはステート4の間には新しい制御信号は何も出力しません。

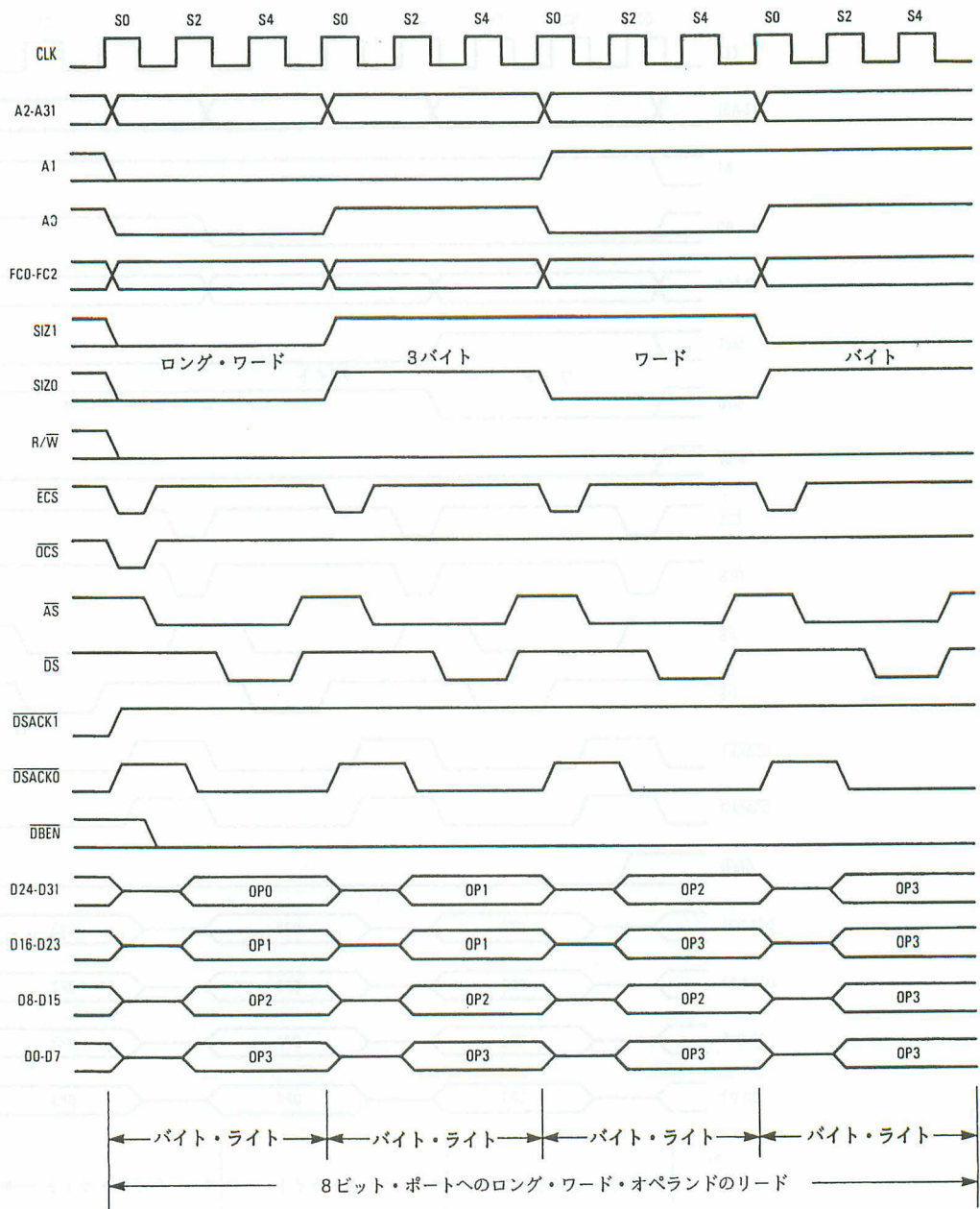



図7-27　ロング・ワード・オペランド・ライト――8ビット・ポート

ステート5

プロセッサはステート5(S5)で$\overline{\mathrm{AS}}$および$\overline{\mathrm{DS}}$をネゲートします。S5の間はアドレスおよびデータ有効を保持し、メモリ・システムにアドレス・ホールド時間を与えます。S5の間、R/$\overline{\mathrm{W}}$、SIZ0およびSIZ1、FC0-FC2、そして$\overline{\mathrm{DBEN}}$も有効になったままです。

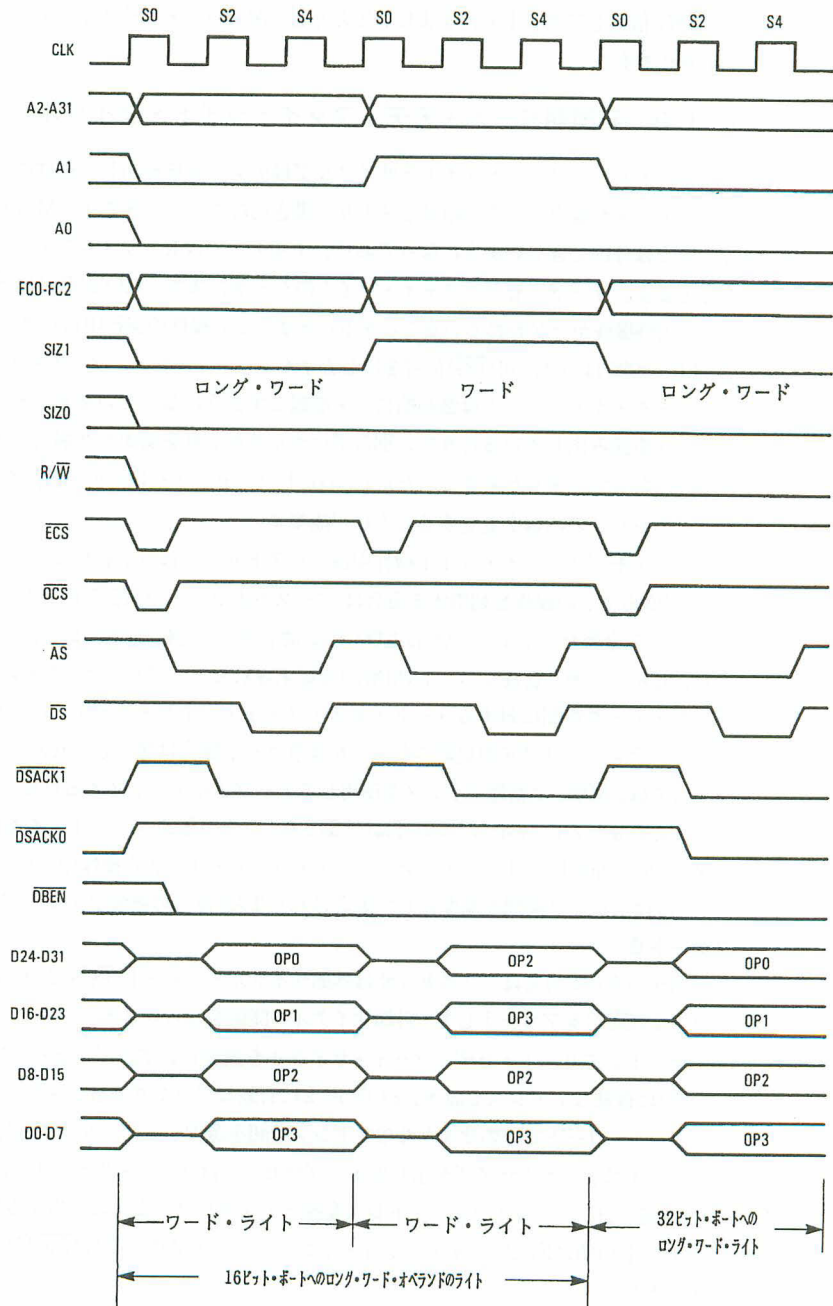



図7-28 ロング・ワード・オペランド・ライト——16ビット・ポート

外部デバイスは、$\overline{\mathrm{AS}}$または$\overline{\mathrm{DS}}$のネゲートを(どちらか最初に)検出するまで、データおよび$\overline{\mathrm{DSACKx}}$信号をアサートしたままにしておきます。デバイスは、$\overline{\mathrm{AS}}$または$\overline{\mathrm{DS}}$のネゲートを検出した後、約1クロック期間以内に$\overline{\mathrm{DSACKx}}$信号をネゲートしなければなりません。この期間を超えて$\overline{\mathrm{DSACKx}}$信号をアサートしたままにしておくと、次のバス・サイクルですぐに検出されてしまうことがあります。

7. 3. 3 非同期リード・モディファイ・ライト・サイクル

リード・モディファイ・ライト・サイクルではデータを読み出し、条件に応じて演算論理ユニット内でデータを変更し、その結果をメモリに書き込むことができます。MC68030プロセッサでは、この操作は分割できないようになっており、マルチ・プロセッサ・システムに対してセマフォ機能を与えます。リード・モディファイ・ライトのシーケンスで、MC68030は$\overline{\mathrm{RMC}}$信号をアサートして、不可分操作が行なわれていることを示します。この操作の実行中にバス要求($\overline{\mathrm{BR}}$)信号があっても、MC68030はバス許可($\overline{\mathrm{BG}}$)信号を出しません。リード・モディファイ・ライト操作のリード部分は、データ・キャッシュでは強制的にミスを起こすようになっています。その理由は、キャッシュのデータを読み出しているときに、別のプロセッサがそれを変更した場合、キャッシュ内のデータが無効になってしまうためです。ただし、「6. 1. 2　データ・キャッシュ」に説明するように、データ・キャッシュの内容を変更することは可能です。

リード・モディファイ・ライト操作中はバースト充てんは行なわれません。

MC68030でこの操作を利用する命令は、“テストとセット(TAS)”命令と“比較と交換(CASおよびCAS2)”命令だけです。CASおよびCAS2命令では、比較結果によってはライト・サイクルは発生しません。メモリ管理ユニット(MMU)で要求されるテーブル・サーチ・アクセスは、常にスーパバイザ・データ空間に対するリード・モディファイ・ライト・サイクルになります。これらのサイクル中、ディスクリプタが更新されないかぎりライト操作は発生しません。テーブル・サーチ・アクセスでは、MMUは物理アドレスを使用してテーブルにアクセスするため、データは内部ではキャッシュされません。MMUについては、「第9章　メモリ管理ユニット」を参照してください。

図7-29に非同期のリード・モディファイ・ライト・サイクル操作のフローチャートを示します。図7-30はクロック期間を基準にして規定されたTAS命令の機能タイミング図です。

ステート0

ステート0(S0)では、プロセッサは外部サイクル・スタート($\overline{\mathrm{ECS}}$)とオペランド・サイクル・スタート($\overline{\mathrm{OCS}}$)をアサートして、外部サイクルの開始を知らせます。プロセッサは、S0では$\overline{\mathrm{RMC}}$もアサートしてリード・モディファイ・サイクルを識別します。プロセッサはアドレス・バス(A0-A31)に有効なアドレスを置き、FC0-FC2に有効なファンクション・コードを置きます。ファンクション・コードでそのサイクルのアドレス空間を選択します。サイズ信号SIZ0とSIZ1が有効となり、オペランドのサイズを示します。プロセッサはリード・サイクルでは、リード/ライト信号R/$\overline{\mathrm{W}}$を“H”にドライブし、アドレス変換ディスクリプタまたは該当するTTxレジスタのMMU CIビットの値に応じて、キャッシュ・インヒビット・アウト信号($\overline{\mathrm{CIOUT}}$)をセットします。

ステート1

半クロック後のステート1(S1)では、プロセッサはアドレス・ストローブ($\overline{\mathrm{AS}}$)をアサートし、アドレス・バス上のアドレスが有効であることを示します。S1では、プロセッサはデータ・ストローブ($\overline{\mathrm{DS}}$)もアサートします。また、S1では$\overline{\mathrm{ECS}}$($\overline{\mathrm{OCS}}$がアサートされているときは$\overline{\mathrm{OCS}}$も)信号がネゲートされます。

ステート2

ステート2(S2)では、プロセッサは$\overline{\mathrm{DBEN}}$をアサートして外部データ・バッファをイネーブルにします。

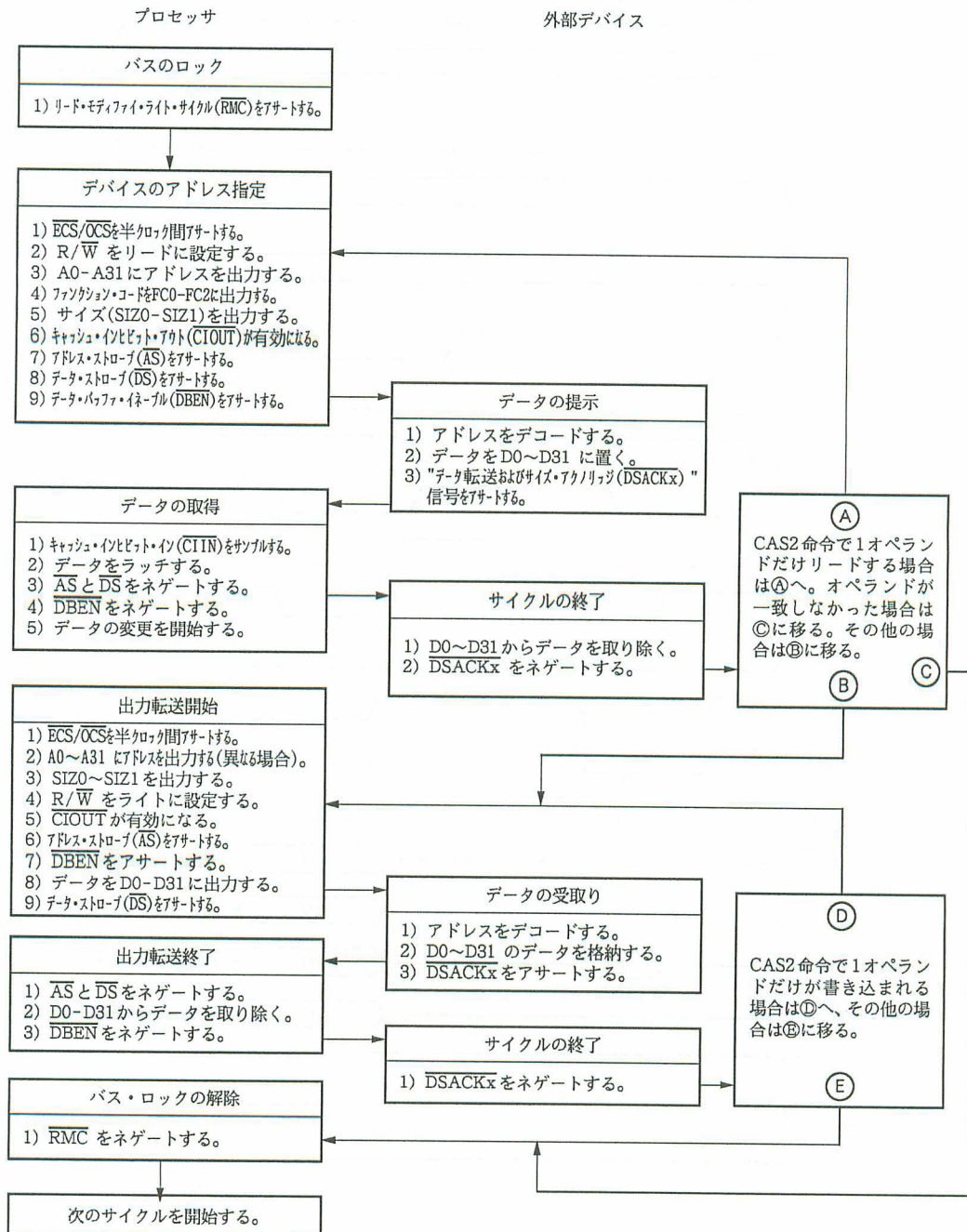



図7-29 非同期リード・モディファイ・ライト・サイクルのフローチャート

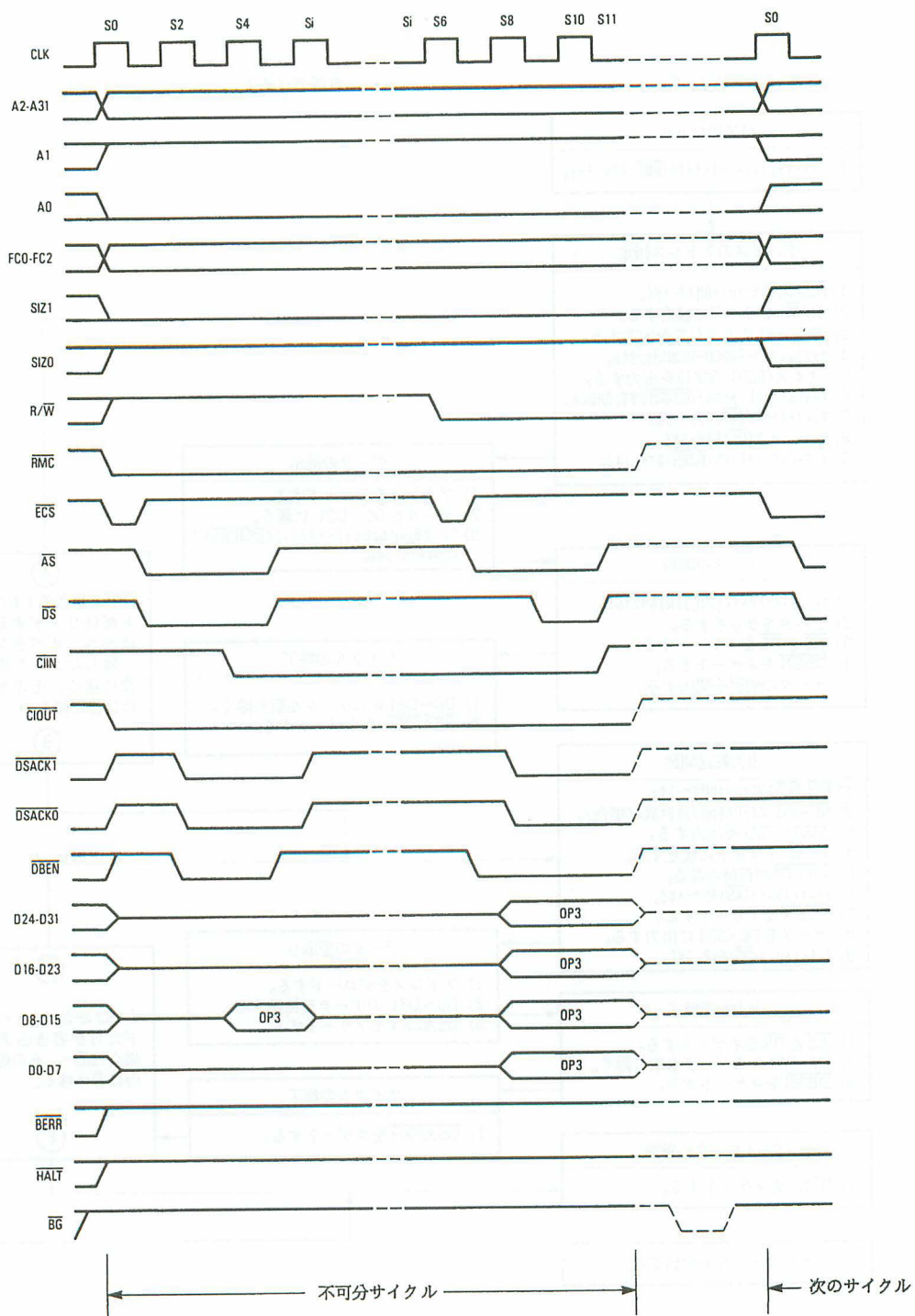



図7-30　非同期バイト・リード・モディファイ・ライト・サイクル――32ビット・ポート（TAS命令、$\overline{\text{CIOUT}}$または$\overline{\text{CINN}}$がアサート）

選択されたデバイスは、R/$\overline{W}$、SIZ0-SIZ1、A0-A1、および$\overline{DS}$を使用してその情報をデータ・バスに置きます。サイズ信号とA0－A1によって、(D24-D31、D16-D23、D8-D15およびD0-D7)の任意またはすべてのバイトが選択されます。同時に、選択されたデバイスはデータ転送およびサイズ・アクノリッジ($\overline{DSACKx}$)信号をアサートすることができます。

ステート3

S2の終わりで、少なくとも1つの$\overline{DSACKx}$信号が認識されると(非同期の入力セットアップ条件を満たすもの)、クロックの次の立下りエッジでデータがラッチされ、サイクルが終了します。ステート3(S3)の初めに$\overline{DSACKx}$が認識されなかった場合、プロセッサはステート4および5に進まずに、ウエイト・ステートを挿入します。ウエイト・ステートを挿入するには、$\overline{DSACK0}$と$\overline{DSACK1}$の両方が、S2の終了付近で非同期入力のセットアップおよびホールド時間の間ネゲートされたままでなければなりません。ウエイト・ステートが追加されると、プロセッサは$\overline{DSACKx}$信号の1つを認識するまで、クロックの立下りエッジで継続してサンプリングを行ないます。

ステート4

プロセッサはステート4(S4)の初めに$\overline{CIIN}$をサンプリングします。S4の終わりで、プロセッサは入力データをラッチします。

ステート5

プロセッサはステート5(S5)で、$\overline{AS}$、$\overline{DS}$、および$\overline{DBEN}$をネゲートします。オペランドを読み出すのに2回以上のリード・サイクルが必要な場合は、各リード・サイクルごとにS0〜S5を繰り返します。読出しを終えると、プロセッサはそのサイクルのライト部分に備えて、アドレス、R/$\overline{W}$、およびFC0-FC2有効を保持します。

外部デバイスは、$\overline{AS}$または$\overline{DS}$のネゲートを(どちらか最初に)検出するまで、データおよび$\overline{DSACKx}$信号をアサートしたままにしておきます。デバイスは、$\overline{AS}$または$\overline{DS}$のネゲートを検出した後、約1クロック期間以内に、データを取り除き、$\overline{DSACKx}$信号をネゲートしなければなりません。この期間を超えてアサートしたままにしておくと、次のバス・サイクルですぐに検出されてしまうことがあります。

アイドル・ステート

アイドル・ステート中、プロセッサは新しい制御信号はアサートしませんが、この時点で内部的にサイクルの変更部分を開始することがあります。ライト・サイクルが不要の場合、ステート6〜11は省略されます。ライト・サイクルが必要な場合、R/$\overline{W}$信号はステート6までリード・モードになったままで、サイクルの前のリード部分との間で、バスが競合しないようにしています。データ・バスはステート8までドライブされません。

ステート6

ステート6(S6)では、プロセッサは$\overline{ECS}$と$\overline{OCS}$をアサートして、別の外部サイクルの開始を知らせます。プロセッサはR/$\overline{W}$をライト・サイクルを示す“L”にドライブします。$\overline{CIOUT}$も有効となり、アドレス変換ディスクリプタまたは該当するTTxレジスタのMMU CIビットの状態を示します。実行するライト操作に応じて、ステート6の間にアドレス・ラインが変化することもあります。

ステート7

ステート7(S7)では、プロセッサはアドレス・ストローブ($\overline{AS}$)をアサートし、アドレス・バス上のアドレスが有効であることを示します。S7では、プロセッサはデータ・バッファ・イネーブル($\overline{DBEN}$)もアサートし、これを使用してデータ・バッファをイネーブルすることができます。また、S7では$\overline{ECS}$(および$\overline{OCS}$がアサートされている場合は$\overline{OCS}$)信号がネゲートされます。

ステート8

ステート8(S8)では、プロセッサは書き込むデータをデータ・バス(D0-D31)に置きます。

ステート9

プロセッサはステート9(S9)で、データ・ストローブ($\overline{\text{DS}}$)をアサートします。この信号は、データ・バス上でデータが安定していることを示します。S8の終わりで、少なくとも1つの$\overline{\text{DSACKx}}$信号が認識されると(非同期の入力セットアップ条件を満たすもの)、1クロック後にこのサイクルが終了します。ステート9(S9)の初めに$\overline{\text{DSACKx}}$が認識されない場合、プロセッサはステート10および11に進まずに、ウエイト・ステートを挿入します。ウエイト・ステートを挿入するには、$\overline{\text{DSACK0}}$と$\overline{\text{DSACK1}}$の両方が、S8の終了付近で非同期入力のセットアップおよびホールド時間の間ネゲートされたままになっていなければなりません。ウエイト・ステートが追加されると、プロセッサは$\overline{\text{DSACKx}}$信号の1つを認識するまで、クロックの立下りエッジで継続してサンプリングを行ないます。

選択されたデバイスは、R/$\overline{\text{W}}$、$\overline{\text{DS}}$、SIZ0-SIZ1、およびA0-A1を使用して、データ・バス(D24-D31、D16-D23、D8-D15、およびD0-D7)の該当するセクションからのデータをラッチします。サイズ信号とA0-A1でデータ・バスのセクションを選択します。デバイスがデータを格納したときに、まだ$\overline{\text{DSACKx}}$がアサートされていなければそれをアサートして、正常にデータを格納したことを知らせます。

ステート10

プロセッサはステート10(S10)の間には新しい制御信号は何も出力しません。

ステート11

プロセッサはステート11(S11)で$\overline{\text{AS}}$と$\overline{\text{DS}}$をネゲートします。S11の間はアドレスおよびデータを有効のまま保持し、メモリ・システムにアドレス・ホールド時間を与えます。S11の間、R/$\overline{\text{W}}$およびFC0-FC2も有効になったままです。

ライト・サイクルが2回以上必要な場合は、ステートS6～S11が各ライト・サイクルごとに繰り返されます。

外部デバイスは、$\overline{\text{AS}}$または$\overline{\text{DS}}$のネゲートを(どちらか最初に)検出するまで、データおよび$\overline{\text{DSACKx}}$信号をアサートしたままにします。デバイスはそのデータを取り除き、$\overline{\text{AS}}$または$\overline{\text{DS}}$のネゲートを検出した後、約1クロック期間以内に、$\overline{\text{DSACKx}}$信号をネゲートしなければなりません。

7.3.4 同期リード・サイクル

同期リード・サイクルは非同期リード・サイクルとは終了方法が異なります。それ以外は、これらのサイクルは同じシーケンスで同じ信号をアサートし、同じ信号に応答します。アドレス指定されたデバイスは、$\overline{\text{DSACKx}}$ではなく同期ターミネーション信号($\overline{\text{STERM}}$)をアサートして、同期リード・サイクルを終了します。$\overline{\text{STERM}}$は$\overline{\text{AS}}$がアサートされている間にクロックのすべての立上りエッジに対して、同期セットアップおよびホールド時間が満たされなければなりません。したがって、プロセッサで同期化を行なう必要はありません。32ビット・ポートのデバイスだけが$\overline{\text{STERM}}$をアサートできます。$\overline{\text{STERM}}$はバースト・モード操作のときに、キャッシュ・バースト要求信号($\overline{\text{CBREQ}}$)とキャッシュ・バースト・アクノリッジ信号($\overline{\text{CBACK}}$)とともに使用することもできます。この信号は32ビット・ポートに対して、2クロック(最小)バス・サイクルと単一クロック(最小)バースト・アクセスを可能にします。これらのサイクルにもウエイト・ステートを挿入することができます。

したがって、同期サイクルは1ウエイト・サイクルの$\overline{\text{STERM}}$で終了すると3クロック・バス・サイクルになります。しかし、$\overline{\text{STERM}}$は$\overline{\text{DSACKx}}$より半クロック後にアサートされるため、ゼロ・ウエイト・サイクル(同じく3クロック)の非同期サイクルに類似しています。したがって、ダイナミック・バス・サイジングが必要ない場合、$\overline{\text{STERM}}$を使用すれば、$\overline{\text{DSACKx}}$による3クロック・ア

クセスより、外部キャッシュ設計に時間的な余裕をもたせることができます。

図7-31に同期ロング・ワード・リード・サイクルのフローチャートを示します。バイト操作とワード操作はよく似ています。図7-32は同期ロング・ワード・リード・サイクルの機能タイミング図です。

ステート0

リード・サイクルはステート0(S0)から始まります。プロセッサは外部サイクル・スタート($\overline{ECS}$)を“L”にドライブして、外部サイクルの開始を知らせます。このサイクルがリード操作での最初の外部サイクルのときは、同時にオペランド・サイクル・スタート($\overline{OCS}$)も“L”にドライブします。S0の間、プロセッサはアドレス・バス(A0-A31)に有効なアドレスを置き、FC0-FC2に有効なファンクション・コードを置きます。ファンクション・コードでそのサイクルのアドレス空間を選択します。プロセッサはリード・サイクルの間、リード/ライト(R/$\overline{W}$)を“H”にドライブし、データ・バッファ・イネーブル($\overline{DBEN}$)を非アクティブにして、データ・バッファをディセーブルします。サイズ信号SIZ0とSIZ1が有効となり、転送に必要なバイト数を示します。キャッシュ・インヒビット・アウト($\overline{CIOUT}$)も有効となり、アドレス変換ディスクリプタまたは該当するTTxレジスタのMMU CIビットの状態を示します。

ステート1

半クロック後のステート1(S1)では、プロセッサはアドレス・ストローブ($\overline{AS}$)をアサートし、アドレス・バス上のアドレスが有効であることを示します。S1では、プロセッサはデータ・ストローブ($\overline{DS}$)もアサートします。該当するオンチップ・キャッシュのバースト・モードがイネーブルになっていて、キャッシュ・エントリの4つのロング・ワードすべてが無効(つまり、4つのロング・ワードが読込み可能)の場合は、$\overline{CBREQ}$がアサートされます。また、S1では$\overline{ECS}$(および$\overline{OCS}$がアサートされている場合は$\overline{OCS}$)信号がネゲートされます。

ステート2

選択されたデバイスは、R/$\overline{W}$、SIZ0-SIZ1、A0-A1、および$\overline{CIOUT}$を使用してその情報をデー

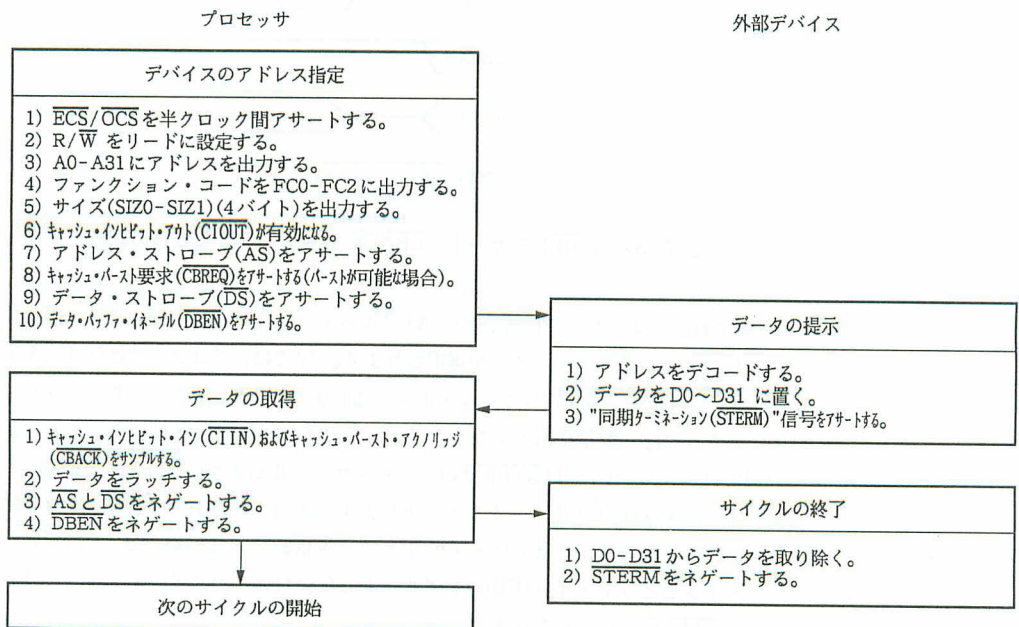



図7-31　同期ロング・ワード・リード・サイクルのフローチャート——バースト不可

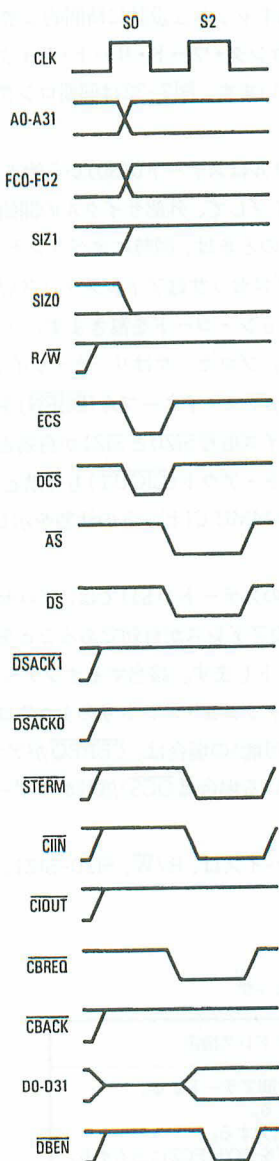



図7-32　$\overline{\text{CIIN}}$をアサート、$\overline{\text{CBACK}}$をネゲートしたときの同期リード

タ・バスに置きます。サイズ信号とA0-A1によって、(D24-D31、D16-D23、D8-D15およびD0-D7)の任意またはすべてのバイトが選択されます。S2では、プロセッサは$\overline{\text{DBEN}}$をアクティブにドライブし、外部データ・バッファをイネーブルします。2クロックの同期バス・サイクルを使用するシステムでは、$\overline{\text{DBEN}}$のタイミングによってはそれが使用できないこともあります。ステート2の初めに、プロセッサは$\overline{\text{STERM}}$のレベルをサンプルします。$\overline{\text{STERM}}$が認識されると、プロセッサはS2の終わりで入力データをラッチします。選択されたデータが現在のサイクルではキャッシュされない場合、またはデバイスが32ビットを供給できない場合は、$\overline{\text{STERM}}$と同時にキャッシュ・インヒビット・イン($\overline{\text{CIIN}}$)をアサートしなければなりません。また、$\overline{\text{STERM}}$が認識されると、$\overline{\text{CBACK}}$がラッチされます。

$\overline{\text{CIIN}}$、$\overline{\text{CBACK}}$および$\overline{\text{STERM}}$は同期信号ですので、$\overline{\text{AS}}$がアサートされている間は、クロックのすべての立上りエッジに対して、同期入力セットアップおよびホールド時間が満たされていな

ければなりません。S2の初めに$\overline{\mathrm{STERM}}$がネゲートされた場合、S2の後にウエイト・ステートが挿入され、その後$\overline{\mathrm{STERM}}$が認識されるまで立上りエッジでサンプリングされます。一度$\overline{\mathrm{STERM}}$が認識されると、クロックの次の立下りエッジ(ステート3の開始に相当する)でデータがラッチされます。

ステート3

プロセッサはステート3(S3)で、$\overline{\mathrm{AS}}$、$\overline{\mathrm{DS}}$、および$\overline{\mathrm{DBEN}}$をネゲートします。S3の間はアドレスを有効のまま保持し、メモリ・インタフェースを容易にします。S3の期間中、R/$\overline{\mathrm{W}}$、SIZ1およびSIZ0、そしてFC0-FC2も有効になったままです。

外部デバイスは、S3の初めから同期ホールド時間の間データをアサートしていなければなりません。また、$\overline{\mathrm{STERM}}$をアサートした1クロック以内にデータを取り除き、$\overline{\mathrm{STERM}}$をアサートしてから2クロック以内に$\overline{\mathrm{STERM}}$をネゲートしなければなりません。そうしないと、プロセッサが次のバス・サイクルで、誤って$\overline{\mathrm{STERM}}$を使用するおそれがあります。

7. 3. 5　同期ライト・サイクル

同期ライト・サイクルは非同期ライト・サイクルとは終了方法が異なり、データ・ストローブは使用できないことがあります。それ以外、これらのサイクルは同じシーケンスで、同じ信号をアサートし同じ信号に応答します。外部デバイスによって、同期ターミネーション信号($\overline{\mathrm{STERM}}$)がアサートされると、同期ライト・サイクルが終了します。前の節で説明した$\overline{\mathrm{STERM}}$は、リード・サイクルだけでなくライト・サイクルにも適用されます。

2クロックの同期ライト・サイクルでは$\overline{\mathrm{DS}}$はアサートされないため、クロック(CLK)をデータをラッチするためのタイミング信号に使用することができます。さらに、2クロックの同期バス・サイクルでは、最後に$\overline{\mathrm{AS}}$をアサートしてから要求される$\overline{\mathrm{STERM}}$をアサートするまで時間がありません。システムは$\overline{\mathrm{AS}}$のアサートでメモリ・ライトの条件付けを行ない、MC68030の内部状態によってライトがアボートされないようにしなければなりません。

図7-33に同期ライト・サイクルのフローチャートを示します。図7-34はウエイト・ステート付き操作の機能タイミング図です。

ステート0

ライト・サイクルはステート0(S0)から始まります。プロセッサは外部サイクル・スタート($\overline{\mathrm{ECS}}$)を“L”にドライブして、外部サイクルの開始を知らせます。このサイクルがライト操作での最初の外部サイクルのときは、同時にオペランド・サイクル・スタート($\overline{\mathrm{OCS}}$)も“L”にドライブします。S0の間、プロセッサはアドレス・バス(A0-A31)に有効なアドレスを置き、FC0-FC2に有効なファンクション・コードを置きます。ファンクション・コードでそのサイクルのアドレス空間を選択します。プロセッサはライト・サイクルの間、リード/ライト(R/$\overline{\mathrm{W}}$)を“L”にドライブします。サイズ信号SIZ0とSIZ1が有効となり、転送するバイト数を示します。キャッシュ・インヒビット・アウト($\overline{\mathrm{CIOUT}}$)も有効となり、アドレス変換ディスクリプタまたは該当するTTxレジスタのMMU CIビットの状態を示します。

ステート1

半クロック後のステート1(S1)では、プロセッサはアドレス・ストローブ($\overline{\mathrm{AS}}$)をアサートし、アドレス・バス上のアドレスが有効であることを示します。S1では、プロセッサはデータ・バッファ・イネーブル($\overline{\mathrm{DBEN}}$)もアサートしますので、これを外部データ・バッファのイネーブルに使用できます。また、S1では$\overline{\mathrm{ECS}}$(および$\overline{\mathrm{OCS}}$がアサートされている場合は$\overline{\mathrm{OCS}}$)信号がネゲートされます。

ステート2

ステート2(S2)では、プロセッサは書き込むデータをデータ・バス(D0-D31)に置きます。

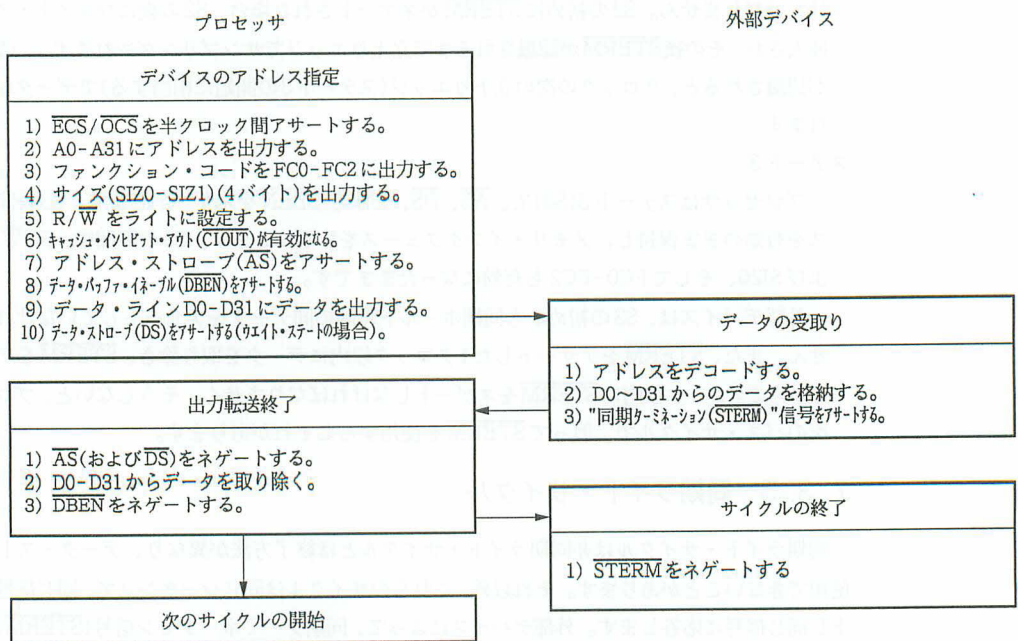



図7-33　非同期ライト・サイクルのフローチャート

選択されたデバイスは、R/$\overline{W}$、SIZ0-SIZ1、およびA0-A1を使用して、データ・バス(D24-D31、D16-D23、D8-D15およびD0-D7)の該当するセクションからのデータをラッチします。サイズ信号とA0－A1でデータ・バス・セクションを選択します。デバイスは正常にデータを格納すると$\overline{STERM}$をアサートします。デバイスがS2の立上りエッジで$\overline{STERM}$をアサートしなかった場合、プロセッサはそれが認識されるまで、ウエイト・ステートを挿入します。ウエイト・ステートを挿入した場合、プロセッサはS2の終わりでデータ・ストローブ($\overline{DS}$)をアサートします。ただし、ゼロ・ウエイト・ステート非同期ライト・サイクルでは、$\overline{DS}$はアサートされません。

ステート3

プロセッサはステート3(S3)で、$\overline{AS}$(および必要なら$\overline{DS}$)をネゲートします。S3の間は、メモリ・インタフェースを容易にするためにアドレスおよびデータ有効を保持しています。S3の期間中、R/$\overline{W}$、SIZ1およびSIZ0、FC0-FC2および$\overline{DBEN}$も有効になったままです。

アドレス指定されたデバイスは、$\overline{STERM}$をアサートしてから2クロック以内にそれをネゲートしなければなりません。そうしないと、プロセッサが次のバス・サイクルで、誤って$\overline{STERM}$を使用するおそれがあります。

7.3.6　同期リード・モディファイ・ライト・サイクル

同期リード・モディファイ・ライト・サイクル操作は、リードおよびライト・サイクルの終了信号とデータのラッチにデータ・ストローブではなくライト・サイクルでのクロック(CLK)を使用する点だけが、非同期リード・モディファイ・ライト・サイクル操作と異なります。

非同期操作と同様、同期リード・モディファイ・ライト・サイクル操作も不可分になっています。操作は同期式ですが、リード・モディファイ・ライト・サイクル中は、バースト・モードを使用してはなりません。

図7-35に同期リード・モディファイ・ライト・サイクル操作のフローチャートを示します。このサイクルのタイミングを図7-36に示します。

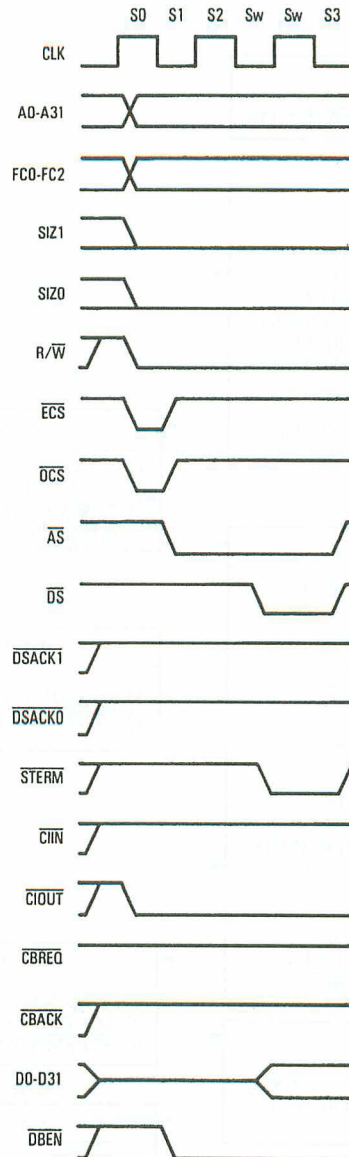



図7-34 $\overline{\text{CIOUT}}$をアサートしたときのウエイト・ステート付き同期ライト・サイクル

ステート0

ステート0では、プロセッサは外部サイクル・スタート($\overline{\text{ECS}}$)とオペランド・サイクル・スタート($\overline{\text{OCS}}$)をアサートして、外部サイクルの開始を知らせます。プロセッサは、S0では$\overline{\text{RMC}}$もアサートしてリード・モディファイ・サイクルを識別します。プロセッサはアドレス・バス(A0-A31)に有効なアドレスを置き、FC0-FC2に有効なファンクション・コードを置きます。ファンクション・コードでそのサイクルのアドレス空間を選択します。サイズ信号SIZ0とSIZ1が有効となり、オペランドのサイズを示します。

プロセッサはリード・サイクルでは、リード/ライト信号R/$\overline{\text{W}}$を“H”にドライブし、アドレス変換ディスクリプタまたは該当するTTxレジスタのMMU CIビットの値に応じて、キャッシ

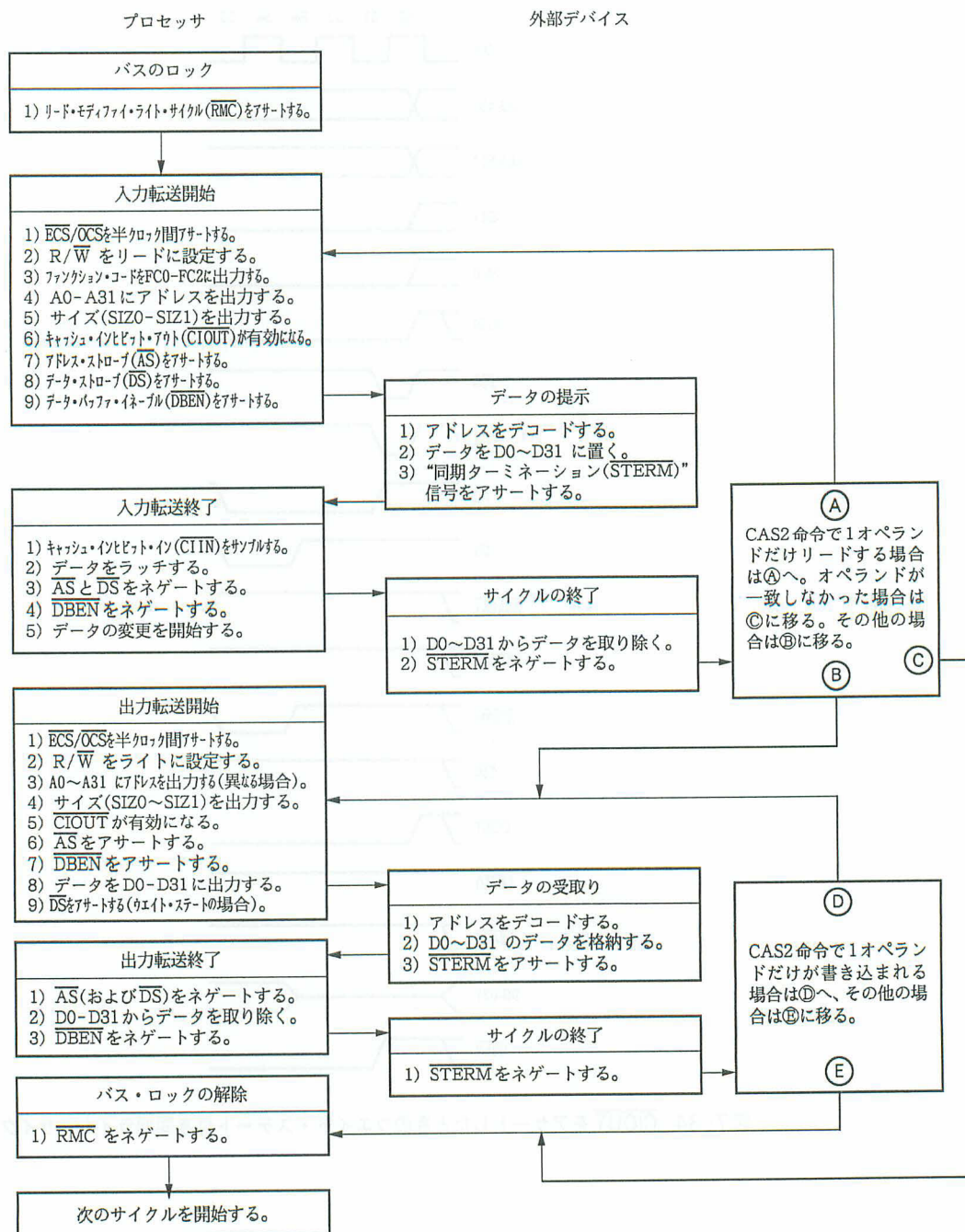



図7-35　同期リード・モディファイ・ライト・サイクルのフローチャート

ュ・インヒビット・アウト信号(CIOUT)をセットします。プロセッサはデータ・バッファ・イネーブル(DBEN)を非アクティブにしてデータ・バッファをディセーブルします。

ステート1

半クロック後のステート1(S1)では、プロセッサはアドレス・ストローブ(AS)をアサートし、ア

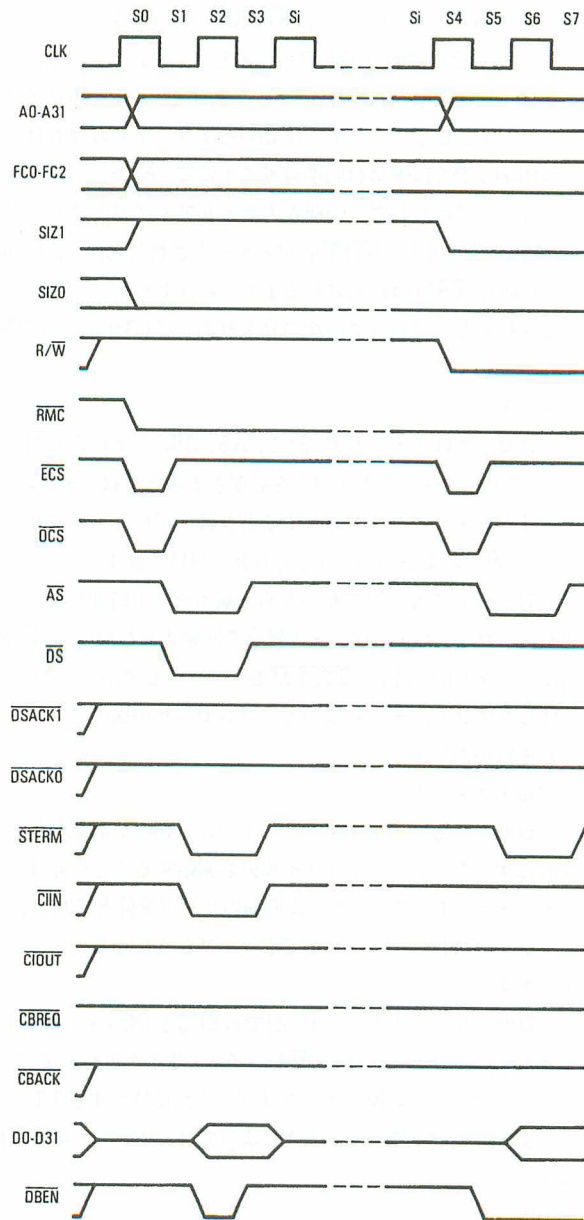



図7-36　$\overline{CIIN}$をアサートしたときの同期リード・モディファイ・ライト・サイクルのタイミング

ドレス・バス上のアドレスが有効であることを示します。S1では、プロセッサはデータ・ストローブ($\overline{DS}$)もアサートします。また、S1では$\overline{ECS}$(および$\overline{OCS}$がアサートされている場合は$\overline{OCS}$)信号がネゲートされます。

ステート2

選択されたデバイスは、R/$\overline{W}$、SIZ0-SIZ1、A0-A1、および$\overline{CIOUT}$を使用してその情報をデータ・バスに置きます。サイズ信号とA0-A1によって、(D24-D31、D16-D23、D8-D15およびD0-D7)の任意またはすべてのバイトが選択されます。S2では、プロセッサは$\overline{DBEN}$をドライブしてアクティブにし、外部データ・バッファをイネーブルします。2クロックの同期バス・サイクルを使用するシステムでは、タイミングによっては$\overline{DBEN}$が使用できないことがあります。ステート

2(S2)の初めに、プロセッサは$\overline{\mathrm{STERM}}$のレベルをサンプルします。$\overline{\mathrm{STERM}}$が認識されると、プロセッサはS2の終わりで入力データをラッチします。選択されたデータが現在のサイクルではキャッシュしない場合、またはデバイスが32ビットを供給できない場合は、$\overline{\mathrm{STERM}}$と同時にキャッシュ・インヒビット・イン($\overline{\mathrm{CIIN}}$)をアサートしなければなりません。

$\overline{\mathrm{CIIN}}$および$\overline{\mathrm{STERM}}$は同期信号ですので、$\overline{\mathrm{AS}}$がアサートされている間は、クロックのすべての立上りエッジに対して、同期入力セットアップおよびホールド時間が満たされていなければなりません。S2の初めに$\overline{\mathrm{STERM}}$がネゲートされた場合、S2の後にウエイト・ステートが挿入され、それ以降$\overline{\mathrm{STERM}}$は認識されるまで、各立上りエッジでサンプリングされます。一度$\overline{\mathrm{STERM}}$が認識されると、クロックの次の立下りエッジ(ステート3の開始に相当)でデータがラッチされます。

ステート3

プロセッサはステート3(S3)で、$\overline{\mathrm{AS}}$、$\overline{\mathrm{DS}}$、および$\overline{\mathrm{DBEN}}$をネゲートします。オペランドの読出しに、2リード・サイクル以上を必要とするときは、それに応じてステートS0～S3を繰り返します。リード・サイクルで終了するときは、プロセッサはサイクルのライト部分に備えて、アドレス、R/$\overline{\mathrm{W}}$、およびFC0-FC2を有効に保持します。

外部デバイスは、S3の初めから同期ホールド時間の間データをアサートしていなければなりません。また、デバイスはバスの競合を回避するために、$\overline{\mathrm{STERM}}$をアサートしてから1クロック以内にデータを取り除き、$\overline{\mathrm{STERM}}$をアサートしてから2クロック以内に$\overline{\mathrm{STERM}}$をネゲートしなければなりません。そうしないと、プロセッサが次のバス・サイクルで、誤って$\overline{\mathrm{STERM}}$を使用するおそれがあります。

アイドル・ステート

プロセッサはアイドル・ステート中は、新しい制御信号をアサートしませんが、この時点で内部的にサイクルのモディファイ部分を開始することがあります。R/$\overline{\mathrm{W}}$信号はステート4までリード・モードのままで、サイクルの前のリード部分との間で、バスが競合しないようにしています。データ・バスはステート6までドライブされません。

ステート4

プロセッサは、ステート4(S4)では$\overline{\mathrm{ECS}}$と$\overline{\mathrm{OCS}}$をアサートして、別の外部サイクルの開始を知らせます。プロセッサはR/$\overline{\mathrm{W}}$をライト・サイクルを示す“L”にドライブします。$\overline{\mathrm{CIOUT}}$も有効となり、アドレス変換ディスクリプタまたは該当するTTxレジスタのMMU CIビットの状態を示します。実行するライト操作によっては、ステート4でアドレス・ラインが変化することもあります。

ステート5

ステート5(S5)では、プロセッサはアドレス・ストローブ($\overline{\mathrm{AS}}$)をアサートし、アドレス・バス上のアドレスが有効であることを示します。S5では、プロセッサはデータ・バッファ・イネーブル($\overline{\mathrm{DBEN}}$)もアサートしますが、これを使用してデータ・バッファをイネーブルすることができます。

ステート6

ステート6(S6)では、プロセッサは書き込むデータをデータ・バス(D0-031)に置きます。

選択されたデバイスは、R/$\overline{\mathrm{W}}$、CLK、SIZ0-SIZ1、およびA0-A1を使用して、データ・バス(D24-D31、D16-D23、D8-D15およびD0-D7)の該当するセクションからのデータをラッチします。サイズ信号とA0-A1でデータ・バス・セクションを選択します。デバイスは正常にデータを格納すると$\overline{\mathrm{STERM}}$をアサートします。デバイスがS2の立上りエッジで$\overline{\mathrm{STERM}}$をアサートしなかった場合、プロセッサはそれが認識されるまで、ウエイト・ステートを挿入します。ウエイト・ステートを挿入した場合、プロセッサはS6の終わりでデータ・ストローブ($\overline{\mathrm{DS}}$)をアサートします。ただし、ゼロ・ウエイト・ステートの非同期ライト・サイクルでは、$\overline{\mathrm{DS}}$はアサートされま

せん。

ステート7

プロセッサはステート7(S7)で、$\overline{AS}$(そして必要なら$\overline{DS}$)をネゲートします。S7の間は、メモリ・インタフェースを容易にするためにアドレスおよびデータ有効を保持します。S7の間は、R/$\overline{W}$およびFC0-FC2は有効になったままです。

2回以上ライト・サイクルが必要な場合は、各ライト・サイクルごとにステートS8〜S11を繰り返します。

外部デバイスは、$\overline{STERM}$をアサートしてから2クロック以内にそれをネゲートしなければなりません。そうしないと、プロセッサが次のバス・サイクルで、誤って$\overline{STERM}$を使用するおそれがあります。

7.3.7　バースト操作サイクル

MC68030は内蔵する命令キャッシュおよびデータ・キャッシュを充てんするのに、バースト・モードをサポートしています。

MC68030はバースト・モードのためのハンドシェイク制御信号群を備えています。キャッシュの1つでミスが発生すると、MC68030はバス・サイクルを開始して、必要なデータまたは命令ストリーム・フェッチを取得します。データまたは命令をキャッシュできる場合には、MC68030はキャッシュ・エントリの充てんを試みます。データ・アクセスのアラインメント状態によっては、2つのキャッシュ・エントリの充てんを試みることがあります。また、プロセッサはキャッシュ・バースト要求信号($\overline{CBREQ}$)をアサートして、バースト充てん操作を要求することもできます。つまり、プロセッサはラインにある複数のエントリを充てんすることができるのです。MC68030は最高4つのロング・ワードのバーストを行なうことができます。

バースト可能なキャッシュ・エントリに対して$\overline{CBREQ}$信号をアサートするメカニズムは、それぞれデータ・キャッシュと命令キャッシュに対応するキャッシュ制御レジスタ(CACR)のデータ・バースト・イネーブル(DBE)ビットと命令バースト・イネーブル(IBE)ビットによってイネーブルされます。次のいずれかの状態があると、MC68030はキャッシュ可能なリード・サイクルに対するキャッシュ・バースト要求を開始(そして、$\overline{CBREQ}$をアサート)します。

- 現在の命令またはデータ・フェッチの論理アドレスとファンクション・コード信号が、それぞれの命令キャッシュまたはデータ・キャッシュのインデックス付きタグ・フィールドに一致しない。
- 該当するキャッシュ内のインデックス付きタグに対応する4つのロング・ワードがすべて無効としてマークされている。

ただし、MC68030はミスアラインメント・アクセスの最初の部分では、残りのアクセスが同じキャッシュ・ラインに対応していない場合は、$\overline{CBREQ}$をアサートしません。詳細については、「6.1.3.1 シングル・エントリ・モード」を参照してください。

適当なキャッシュがイネーブルされていなかったり、キャッシュのキャッシュ凍結ビットがセットされている場合、プロセッサは$\overline{CBREQ}$をアサートしません。$\overline{CBREQ}$はリード・モディファイ・ライト操作およびライト・サイクル中にはアサートされません。

MC68030では、同期ターミネーション信号($\overline{STERM}$)でバス・サイクルを終了し、キャッシュ・バースト・アクノリッジ信号($\overline{CBACK}$)をアサートして$\overline{CBREQ}$に応答する32ビット・ポートからしか、バースト充てんを行なうことはできません。MC68030は、$\overline{STERM}$と$\overline{CBACK}$を認識して$\overline{CBREQ}$をアサートすると、バースト操作の間はアドレス・ストローブ($\overline{AS}$)、データ・ストローブ($\overline{DS}$)、リード/ライト(R/$\overline{W}$)、アドレス・バス(A0-A31)、ファンクション・コード(FC0-FC2)、およびサイズ信号(SIZ0－SIZ1)を現在の状態のままで維持します。プロセッサはバーストが完了するか異常終了するまでは、$\overline{STERM}$がアサートされている間、連続して各クロックごとにデータを受

け入れます。

$\overline{CBACK}$は、アドレス指定されたデバイスが、バースト・モードでさらに1つ以上のロング・ワード・データを供給して、キャッシュ・バースト要求に応答できることを示します。$\overline{CBACK}$は$\overline{CBREQ}$信号とは関係なくアサートでき、同期サイクルではこれらの信号の両方がアサートされた場合にのみバースト・モードが開始されます。MC68030がフル・バースト操作を実行して、4つのロング・ワードをフェッチすると、第3サイクルで$\overline{STERM}$がアサートされたのち、$\overline{CBREQ}$がネゲートされ、MC68030はもう1つだけロング・ワード(第4サイクル)を要求していることを示します。ここで$\overline{CBACK}$をネゲートすることができ、MC68030は第4サイクルのデータをラッチし、キャッシュ・ライン充てん操作を完了します。

バースト充てんは、次のいずれかの状態が発生するとアボートすることがあります。

- ●キャッシュ・インヒビット・イン($\overline{CIIN}$)がアサートされた。
- ●バス・エラー($\overline{BERR}$)がアサートされた。
- ●$\overline{CBACK}$が早くネゲートされた。

バースト充てん操作中のバス・エラーの処理は、「7. 5. 1　バス・エラー」で説明します。プロセッサをホルトさせたり、バス要求($\overline{BR}$)でプロセッサからバスを切り離したりするために、バースト操作はシングル・サイクルとなっています。これはアドレス・ストローブ($\overline{AS}$)が操作中にはアサートされたままになっているためです。バースト操作中に$\overline{HALT}$信号をアサートすると、プロセッサは操作が終わるとホルトします。ホルト操作の詳細については、「7. 5. 3　ホルト操作」を参照してください。$\overline{BR}$でバスを要求している代替バス・マスタは、$\overline{BR}$信号が次のプロセッサ・サイクルが始まる前に、内部で十分に同期がとれるだけの余裕をもってアサートされている場合は、バス・マスタになることができます。バス調停の詳細については、「7. 7　バス調停」を参照してください。

バス・サイクル中に$\overline{BERR}$と$\overline{HALT}$が同時にアサートされたときは、通常サイクルの再試行が必要なことを示しています。しかしながら、バースト操作の第2、第3、または第4サイクルでは、この信号の組合せはバス操作をアボートするバス・エラー状態を示します。また、プロセッサは$\overline{HALT}$がネゲートされるまで、ホルト状態になったままです。バス・エラー処理の詳細は、「7. 5. 1　バス・エラー」を参照してください。

図7-37にバースト操作のフローチャートを示します。次のタイミング図に各種のバースト操作を示します。図7-38に最初のアクセスに2つのウエイト・ステートを挿入し、それ以降のアクセスに1つのウエイト・ステートを挿入するロング・ワードのバースト操作を示します。図7-39に$\overline{CBACK}$が早くネゲートされたために、通常どおり完了できなかったバースト操作を示します。図7-40にオペランド全体が同じキャッシュ・ラインに対応していないため、延期されるバースト操作を示します。図7-41はキャッシュ・インヒビット・イン($\overline{CIIN}$)信号でアボートされるバースト操作を示します。$\overline{CBACK}$は次のサイクルに対応していますので、$\overline{CBACK}$が2クロック期間しかアサートされなくても、3つのロング・ワードが転送されます。

バースト操作シーケンスは、ステート0-3から始まり、$\overline{CBREQ}$がアサートされることを除いて、同期リード・サイクルとほぼ同じです。ステート4-9は完全なバースト操作で、最後の3つのリード・サイクルを実行します。

ステート0

バースト操作はステート0(S0)から始まります。プロセッサは外部サイクル・スタート($\overline{ECS}$)を“L”にドライブして、外部サイクルの開始を知らせます。このサイクルがリード操作での最初の外部サイクルのときは、同時にオペランド・サイクル・スタート($\overline{OCS}$)も“L”にドライブします。S0の間、プロセッサはアドレス・バス(A0-A31)に有効なアドレスを置き、FC0-FC2に有効なファンクション・コードを置きます。ファンクション・コードでそのサイクルのアドレス空間を選択します。プロセッサはリード/ライト(R/$\overline{W}$)を“H”にドライブして、リード・サイクルを示

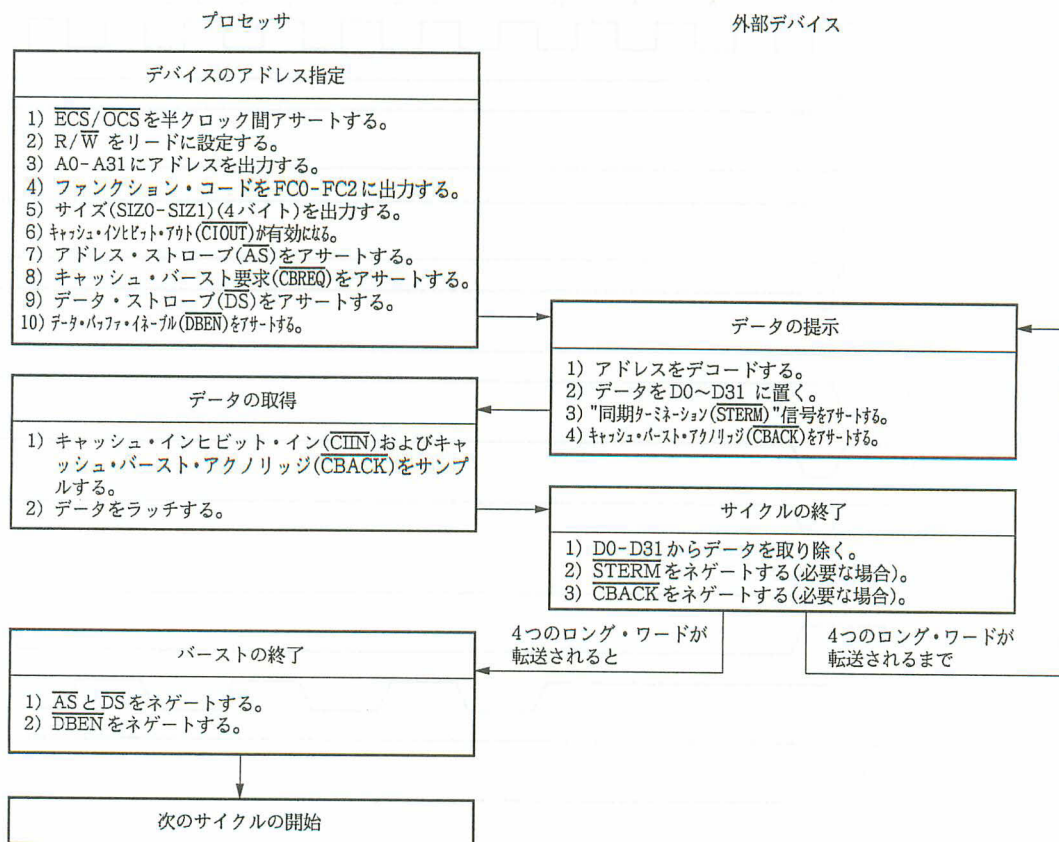



図7-37 バースト操作のフローチャート――4つのロング・ワードの転送

し、データ・バッファ・イネーブル(DBEN)を非アクティブにして、データ・バッファをディセーブルします。サイズ信号SIZ0とSIZ1が有効となり、転送するバイト数を示します。キャッシュ・インヒビット・アウト(CIOUT)も有効となり、アドレス変換ディスクリプタまたは該当するTTxレジスタのMMU CIビットの状態を示します。

ステート1

半クロック後のステート1(S1)では、プロセッサはアドレス・ストローブ(AS)をアサートし、アドレス・バス上のアドレスが有効であることを示します。S1では、プロセッサはデータ・ストローブ(DS)もアサートします。

CBREQもアサートされ、MC68030がそのキャッシュの1つに対してバースト操作を実行し、4つのロング・ワードに読み込むことができることを示します。また、S1ではECS(およびOCSがアサートされている場合はOCS)信号がネゲートされます。

ステート2

選択されたデバイスは、R/W、SIZ0-SIZ1、A0-A1、およびCIOUTを使用してそのデータをデータ・バスに置きます(最初のサイクルは、対応するロング・ワード境界においてロング・ワードを供給しなければなりません)。

バースト操作は各サイクルで32ビットをラッチするため、データ・バスの全部のバイト・セクション(D24-D31、D16-D23、D8-D15およびD0-D7)をドライブしなければなりません。S2では、プロセッサはDBENをアクティブにドライブして、外部データ・バッファをイネーブルし

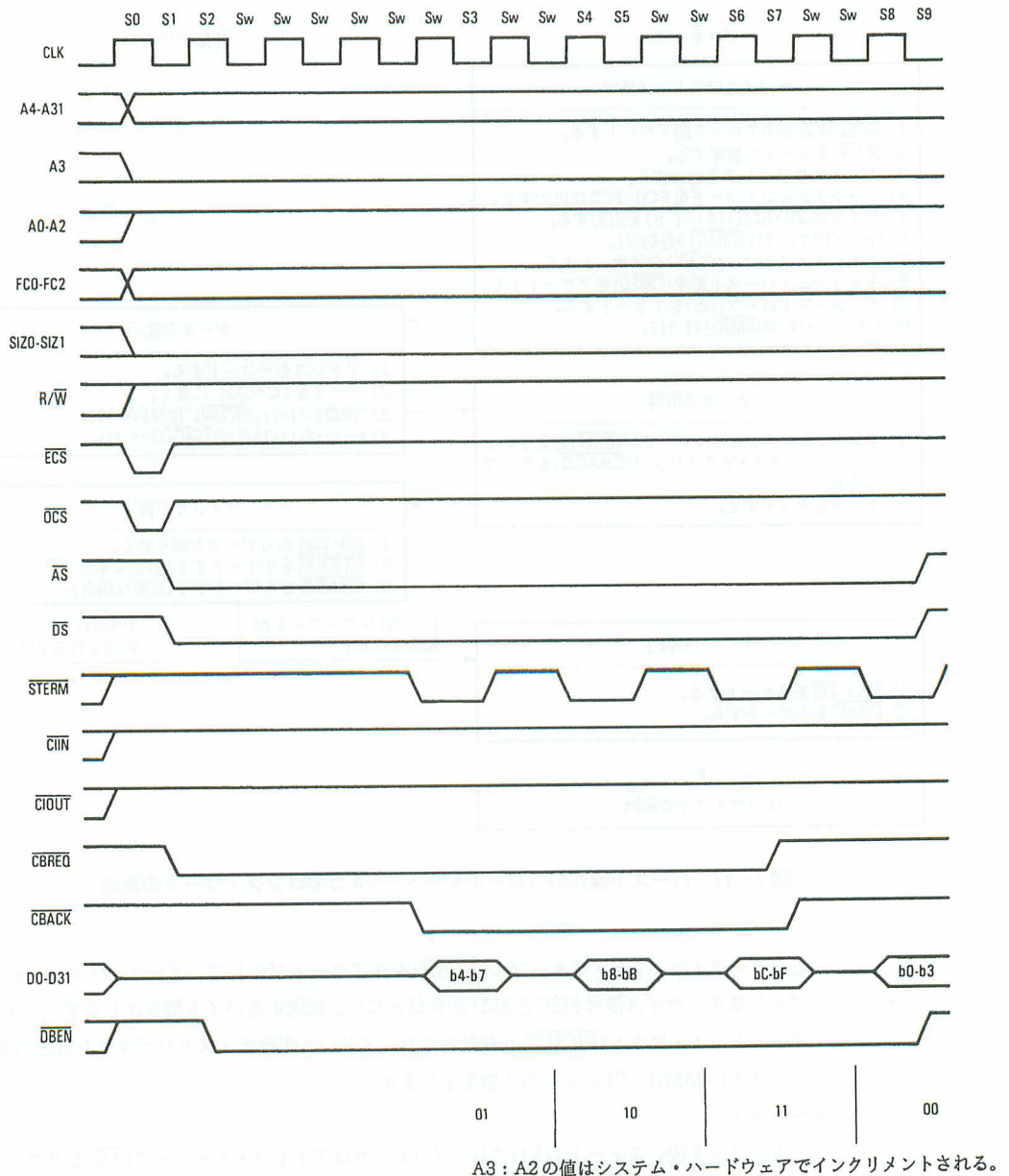



図7-38　バースト要求およびウエイト・サイクルによる＄07からの
ロング・ワード・オペランド要求

ます。2クロックの同期バス・サイクルを使用するシステムでは、タイミングによっては$\overline{\text{DBEN}}$が使用できないことがあります。ステート2(S2)の初めに、プロセッサは$\overline{\text{STERM}}$のレベルをテストします。$\overline{\text{STERM}}$を認識すると、プロセッサはS2の終わりで入力データをラッチします。$\overline{\text{STERM}}$を認識したら、$\overline{\text{CBACK}}$をアサートしてバースト操作を進めなければなりません。現在のサイクルでデータをキャッシュしない場合は、$\overline{\text{STERM}}$と同時にキャッシュ・インヒビット・イン($\overline{\text{CIIN}}$)をアサートしなければなりません。また、$\overline{\text{CIIN}}$をアサートするとバースト操作もアボートされます。

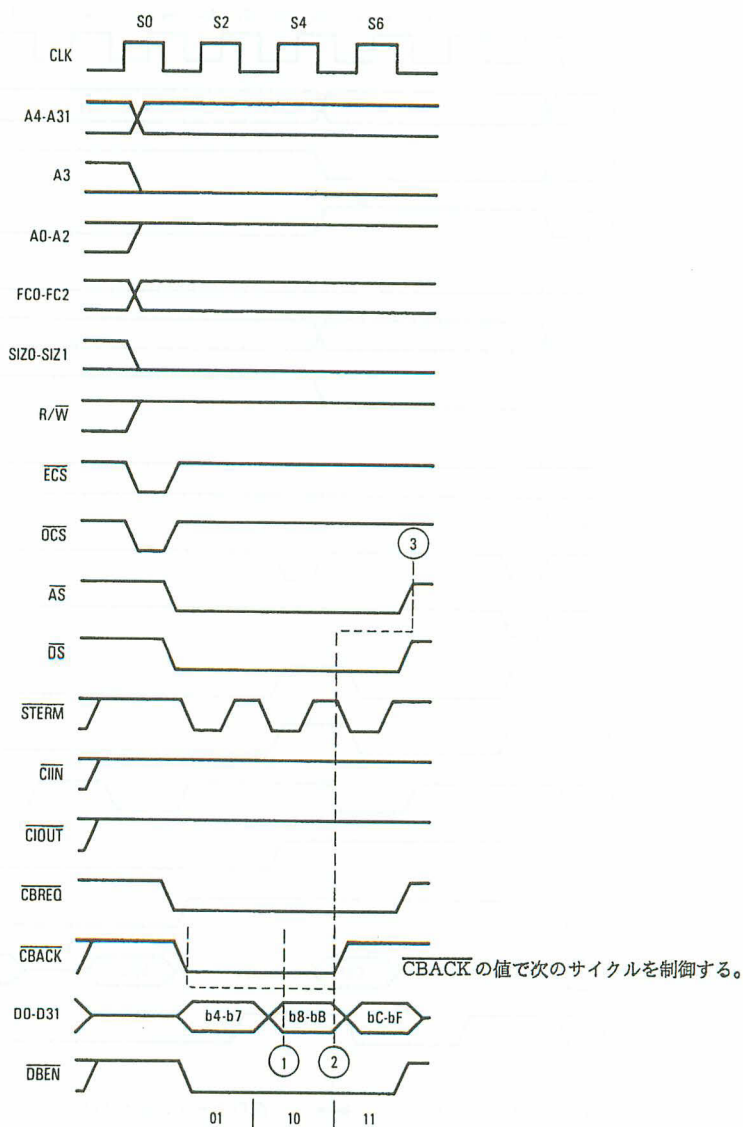
$\overline{\text{CIIN}}$、$\overline{\text{CBACK}}$および$\overline{\text{STERM}}$は同期信号ですので、$\overline{\text{AS}}$がアサートされている間は、クロック

A3：A2の値はシステム・ハードウェアでインクリメントされる。
注：1. CBACKをアサートすると、データがD0-D31に置かれる。
　　2. CBACKを継続してアサートすると、データがD0-D31に置かれる。
　　3. CBACKをネゲートすると、ASがネゲートされる。

図7-39　バースト要求による＄07からのロング・ワード・オペランド要求

のすべての立上りエッジに対し、セットアップおよびホールド時間が満たされていなければなりません。

S2の初めにSTERMがネゲートされた場合は、S2の後にウエイト・ステートが挿入され、それ以降STERMを認識するまで、各立上りエッジでサンプリングが行なわれます。一度STERMが認識されると、クロックの次の立下りエッジ（ステート3の開始に相当）でデータがラッチされます。

ステート3

プロセッサはステート3（S3）の間、AS、DS、およびDBENをアサートしたままにします。S3の間はアドレスを有効のまま保持し、バースト操作を継続します。S3の期間中、R／W、SIZ1と

図7-40　$ 0Eからのロング・ワード・オペランド要求——バースト充てんの延期

SIZ0、およびFC0-FC2も有効になったままです。

外部デバイスは、S3の初めから同期ホールド時間の間データをアサートしていなければなりません。また、$\overline{\text{STERM}}$をアサートしてから1クロック以内に$\overline{\text{STERM}}$をネゲートしなければなりません。そうしないと、プロセッサが次のバス・サイクルで、誤って$\overline{\text{STERM}}$を使用するおそれがあります。$\overline{\text{STERM}}$は後続のアクセスがウエイト・サイクルを必要としない場合は、ネゲートする必要はありません。

ステート4

ステート4(S4)の初めにプロセッサは$\overline{\text{STERM}}$のレベルをテストします。このステートはバースト・モードの開始を知らせ、残りのステートはバースト充てんサイクルになります。$\overline{\text{STERM}}$が認識されると、プロセッサはS4の終わりで入力データをラッチします。このデータはバーストの2番目のロング・ワードに対応します。S4の初めに$\overline{\text{STERM}}$がネゲートされたときは、S4およびS5

図7-41　バースト要求を伴う＄07からのロング・ワード・オペランド要求
——$\overline{\text{CBACK}}$および$\overline{\text{CINN}}$がアサート

の代わりにウエイト・ステートが挿入され、それ以降$\overline{\text{STERM}}$は認識されるまで、各立上りエッジでサンプリングされます。同期サイクルでは、$\overline{\text{STERM}}$が認識されると$\overline{\text{CBACK}}$と$\overline{\text{CIIN}}$の状態がラッチされます。この時点で、$\overline{\text{CBACK}}$をアサートした場合は、バースト操作を継続しなければならないことを示します。$\overline{\text{CIIN}}$をアサートした場合は、S4の終わりでラッチしたデータをキャッシュして、バーストをアボートしなければならないことを示します。

ステート5

プロセッサはステート5の期間中、バス上のすべての信号をドライブしたままにしてバーストを継続します。ここでは、S3で述べた$\overline{STERM}$およびデータのホールド時間が適用されます。

ステート6

このステートは、一度$\overline{STERM}$が認識されると、データの3番目のロング・ワードがS6の終わりでラッチされることを除いてステート4と同じです。

ステート7

このステートでは、プロセッサが$\overline{CBREQ}$信号をネゲートし、メモリ・デバイスが$\overline{CBACK}$をネゲートすることがあります。これとは別に、プロセッサがドライブする他のすべてのバス信号はドライブされたままです。ここでは、S3で述べた$\overline{STERM}$およびデータのホールド時間が適用されます。

ステート8

このステートでは、$\overline{CBREQ}$がネゲートされプロセッサがそれ以上データを受け入れることができないことを示します。それ以外はステート4と同じです。S8の終わりでラッチされたデータは、バーストの4番目のロング・ワードに対応します。

ステート9

プロセッサはS9で、$\overline{AS}$、$\overline{DS}$、および$\overline{DBEN}$をネゲートします。S9ではアドレス、R/W、SIZ0、SIZ1、およびFC0-FC2を有効に保持します。ここでは、S3で述べたのと同じホールド時間条件が適用されます。

MC68030のアドレス・バスは、バースト転送操作中には(バースト・モードに入ったときの最初の転送を含む)、一定の値にドライブされています。連続したロング・ワード情報を供給するために、外部メモリ・システムでロング・ワードのベース・アドレスをインクリメントする必要がある場合は、外部ハードウェアでこの機能を実行しなければなりません。

さらに、16バイト境界をまたがるバースト転送(つまり、最初に転送されたロング・ワードがA3/A2＝00に存在しない)の場合は、必要に応じて、外部ハードウェアがバースト転送の継続または終了を適切に制御しなければなりません。バースト操作は16バイト・イメージ(A3/A2＝11)の最上位ロング・ワードの転送中に$\overline{CBACK}$をネゲートして終了するか、A3/A2＝00に存在するロング・ワードを供給することによって($\overline{CBACK}$をアサートして)、継続することができます(つまり、カウント・シーケンスがゼロにラップ・バックし、必要に応じて継続される)。MC68030のキャッシュは、ロング・ワード・アクセスがA3/A2＝00付近にラップ・バックしても、上位アドレス・ラインA4-A31は変化しないものと仮定しています。

7. 4　CPU 空間サイクル

ファンクション・コード(FC0-FC2)は、ユーザ・プログラム、スーパバイザ・プログラム、およびデータ領域を表4-1に示すとおり選択します。この領域は、ファンクション・コードFC0-FC2＝＄7で選択され、CPU空間として分類されています。以下の項で述べる、割込みアクノリッジ、ブレークポイント・アクノリッジ、およびコプロセッサ通信サイクルはCPU空間を使用します。

CPU空間の種類は、CPU空間操作中にアドレス信号A16-A19にエンコードされ、プロセッサが実行している機能を示します。MC68030では図7-42に示すとおり、3つのエンコーディングが実装されています。未使用の値はすべて、将来の拡張に備えてモトローラが予約しています。

7. 4. 1　割込みアクノリッジ・バス・サイクル

周辺デバイスがプロセッサに対してサービスを要求し($\overline{IPL0}$-$\overline{IPL2}$信号により)、これらの信号を

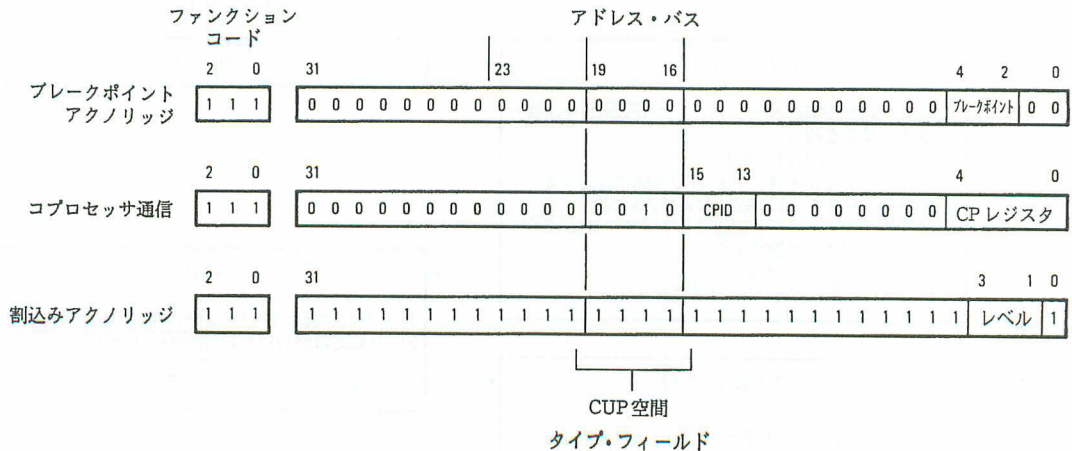



図7-42　MC68030のCPU空間のアドレス・エンコーディング

内部に取り込んだ値がステータス・レジスタの割込みマスクの優先順位よりも高い(あるいは、レベル7割込みの場合は遷移が発生した)場合、プロセッサはその割込みを保留割込みにします。割込みの認識については、「8. 1. 9　割込み例外」を参照してください。

MC68030は保留された割込みに対する割込み例外を(それより高い優先順位をもつ保留された例外を処理した後)、1つの命令の境界で実行します。以下のパラグラフでは、割込み例外処理の一部として実行される、各種割込みアクノリッジ・サイクルを説明します。

7. 4. 1. 1　割込みアクノリッジ・サイクル――通常終了

MC68030が割込み例外を処理するときは、割込みアクノリッジ・サイクルを実行して割込みサービス・ルーチンの開始ロケーションをもつベクタ番号を取得します。

割込みを要求しているデバイスには、使用するルーチンの割込みベクタを保持するプログラム可能なベクタ・レジスタをもっているものもあります。以下のパラグラフでは、これらのデバイスのための割込みアクノリッジ・サイクルについて説明します。他の割込み条件やデバイスは、ベクタ番号を供給できないため、「7. 4. 1. 2　オートベクタ割込みアクノリッジ・サイクル」で説明するオートベクタ・サイクルを使用します。

割込みアクノリッジ・サイクルは、リード・サイクルです。このサイクルは「7. 3. 1　非同期リード・サイクル」で説明する非同期リード・サイクルや、「7. 3. 4　同期リード・サイクル」で説明する同期リード・サイクルとは、CPUアドレス空間をアクセスする点が異なります。主な違いは次のとおりです。

1. ファンクション・コード(FC0-FC2)は、CPUアドレス空間では7(FC0/FC1/FC2＝111)にセットされる。
2. アドレス信号A1、A2、およびA3は割込み要求レベル(それぞれ、$\overline{\mathrm{IPL0}}$、$\overline{\mathrm{IPL1}}$、および$\overline{\mathrm{IPL2}}$の反転した値)にセットされる。
3. CPU空間タイプ・フィールド(アドレス信号A16-A19)は、割込みアクノリッジ・コード＄Fにセットされる。
4. 他のアドレス信号(A20-A31、A4-A15およびA0)は1にセットされる。

応答デバイスは、割込みアクノリッジ・サイクル中にデータ・バスにベクタ番号を置きます。これを超えた場合、サイクルは通常どおり$\overline{\mathrm{STERM}}$または$\overline{\mathrm{DSACKx}}$で終了します。図7-43に割込みアクノリッジ・サイクルのフローチャートを示します。

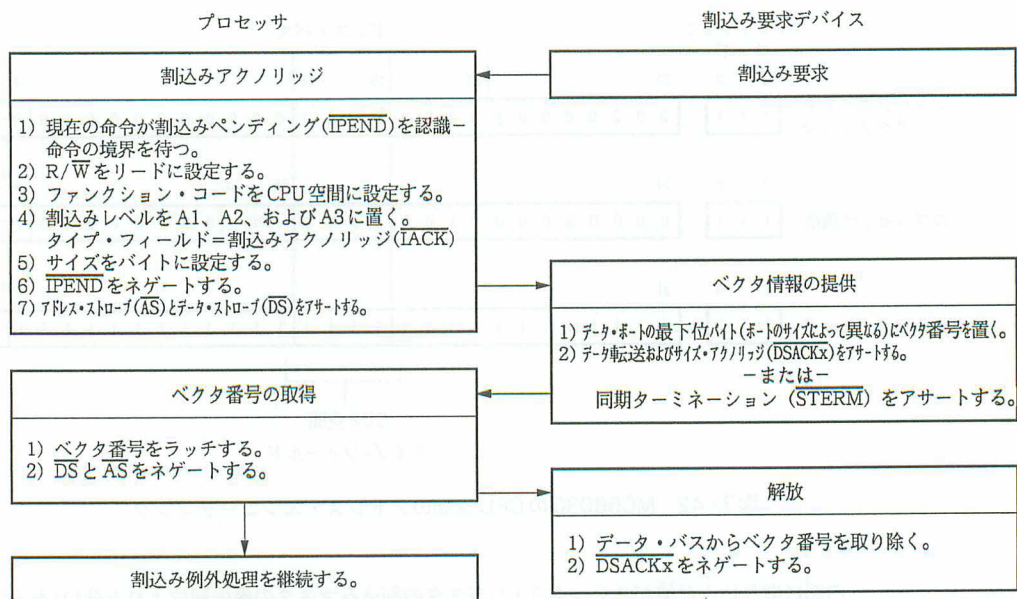



図7-43 割込みアクノリッジ・サイクルのフローチャート

7. 4. 1. 2 オートベクタ割込みアクノリッジ・サイクル

割込みを要求しているデバイスがベクタ番号を供給できないときは、自動的に生成されるベクタ、つまり“オートベクタ”を要求します。データ・バスにベクタ番号を置き、データ転送およびサイズ・アクノリッジ信号($\overline{DSACKx}$)または同期ターミネーション信号($\overline{STERM}$)をアサートする代わりに、デバイスはオートベクタ信号($\overline{AVEC}$)をアサートして、そのサイクルを終了します。$\overline{STERM}$または$\overline{DSACKx}$のどちらも、$\overline{AVEC}$で終了する割込みアクノリッジ・サイクル中はアサートできません。

オートベクタ操作で供給されるベクタ番号は、現在要求を行なっている割込みの割込みレベルから得られます。割込みアクノリッジ・サイクル中に$\overline{DSACK}$または$\overline{STERM}$の代わりに、$\overline{AVEC}$信号がアサートされると、MC68030はデータ・バスの状態を無視して、内部的に割込みレベルに24($18)を加えた値に相当するベクタ番号を生成します。明確に分類された7つのオートベクタがあり、それぞれ信号$\overline{IPL0}$-$\overline{IPL2}$で提供される7つの割込みレベルに対応しています。図7-45にオートベクタ操作のタイミングを示します。

7. 4. 1. 3 スプリアス割込みサイクル

デバイスが割込みアクノリッジ・サイクルに対して、$\overline{AVEC}$、$\overline{STERM}$、または$\overline{DSACKx}$をアサートして応答しないときには、外部ロジック回路は通常バス・エラー信号($\overline{BERR}$)を返します。この場合、MC68030は割込みベクタ番号の代わりに、自動的にスプリアス割込みベクタ番号24を生成します。このときホルト信号$\overline{HALT}$もアサートされていた場合、プロセッサはこのサイクルを再試行します。

7. 4. 2 ブレークポイント・アクノリッジ・サイクル

ブレークポイント命令(BKPT)を実行すると、ブレークポイント・アクノリッジ・サイクルが生成されます。ブレークポイント・アクノリッジ・サイクルにより、外部ハードウェアは、プログラム内で実行されるべき命令ワードを直接命令パイプラインに供給します。このサイクルは、タイプ・フ

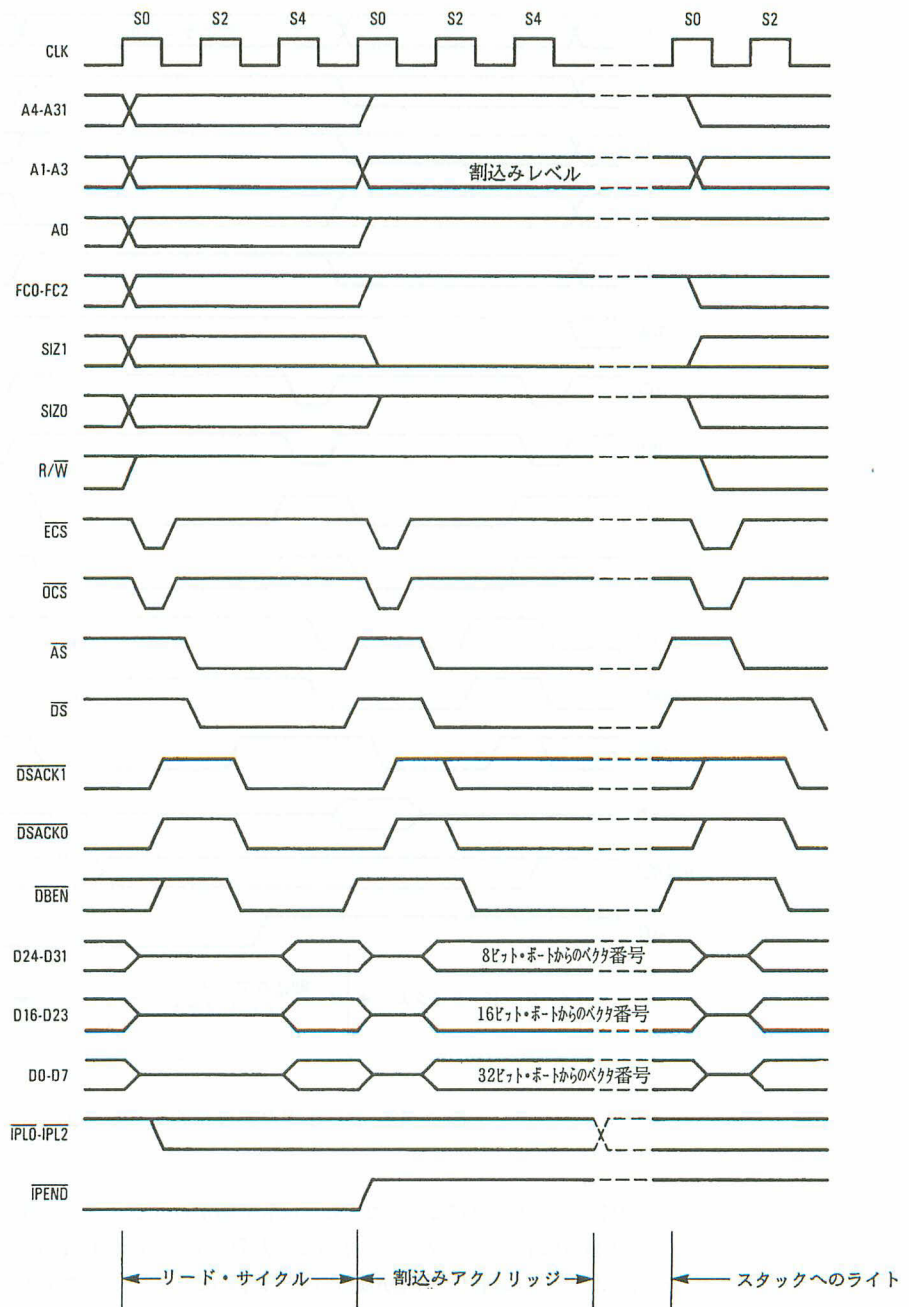



図7-44 割込みアクノリッジ・サイクルのタイミング

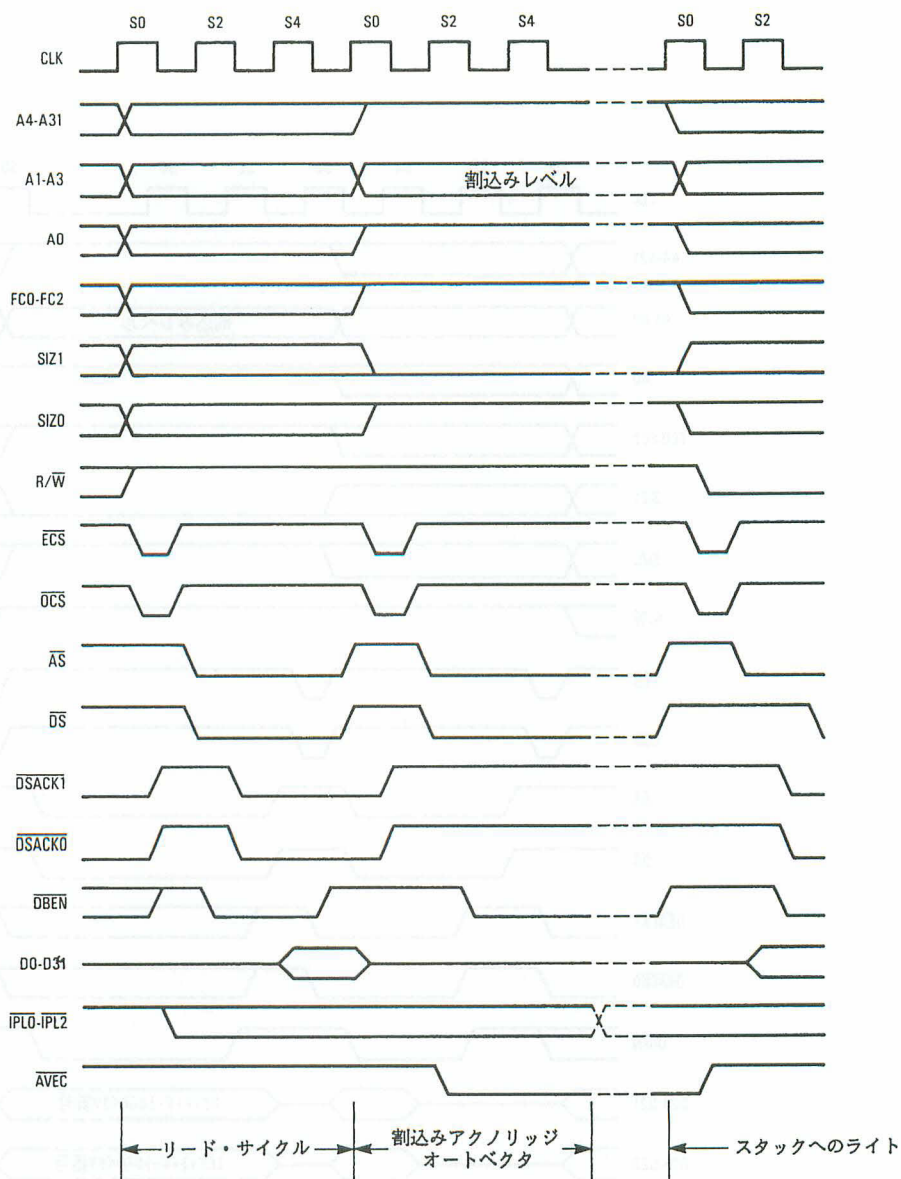



図7-45　オートベクタ操作のタイミング

ィールドを0にしてCPU空間にアクセスし、命令で指定されるブレークポイント番号をアドレス・ラインA2-A4に供給します。外部ハードウェアが$\overline{\text{DSACKx}}$または$\overline{\text{STERM}}$でサイクルを終了した場合、バス上のデータ(命令ワード)が命令パイプに挿入され、ブレークポイント・オペコードを置き換えて、ブレークポイント・アクノリッジ・サイクルが完了した後で実行されます。ブレークポイント命令ではワードを転送する必要があるため、最初のバス・サイクルで8ビット・ポートをアクセスした場合は、第2サイクルが必要です。外部ロジックが$\overline{\text{BERR}}$でブレークポイント・アクノリッジ・サイクルを終了すると(つまり、命令ワードがない)、プロセッサは不当命令例外の処理を行ないます。図7-46はブレークポイント・アクノリッジ・サイクルのフローチャートです。図7-47は命令ワードを返す、ブレークポイント・アクノリッジ・サイクルのタイミングを示します。また、図

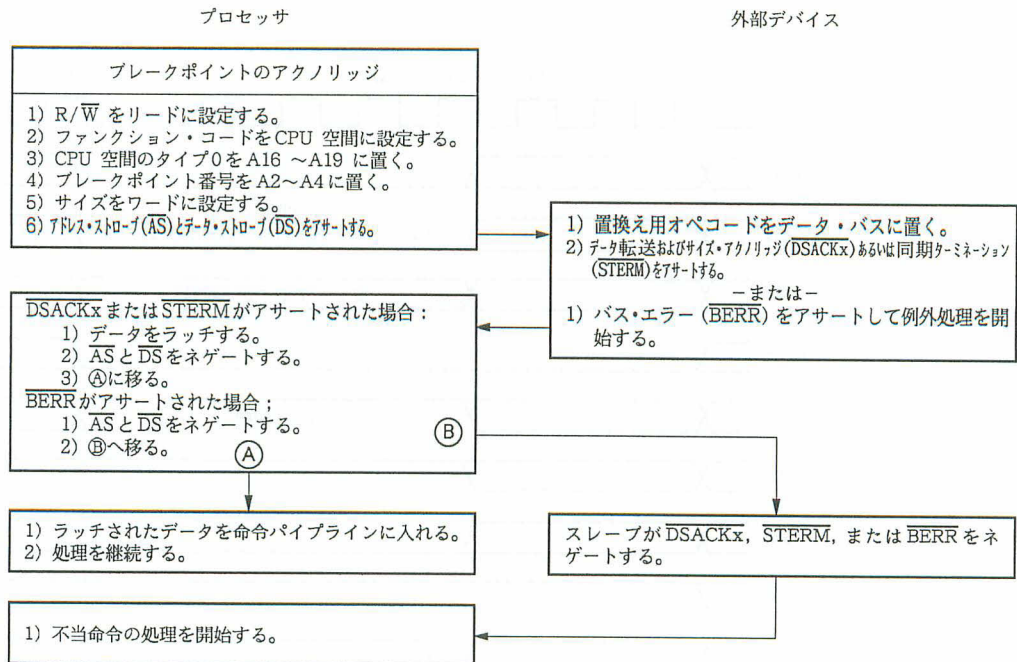



図7-46　ブレークポイト操作のフロー

7-48に例外を発生するブレークポイント・アクノリッジ・サイクルのタイミングを示します。

7. 4. 3　コプロセッサ通信サイクル

MC68030コプロセッサ・インタフェースにより、プロセッサと7個までのコプロセッサ間で、命令を主体とする通信を行なうことができます。コプロセッサとの間の操作に必要なバス通信は、$2のタイプ・フィールドをもつMC68030のCPU空間で実行されます。

コプロセッサのアクセスは、MC68030のバス・プロトコルを使用します。ただし、アドレス・バスは32ビットのアドレスではなくアクセス情報を供給します。コプロセッサ操作のためのCPU空間のタイプ・フィールド(アドレス信号A16 ～A19)は$2です。アドレス信号A13～A15にはコプロセッサの識別番号(Cp-ID)が含まれており、アドレス信号A0～A4はアクセスするコプロセッサ・インタフェース・レジスタを示します。Cp-IDが0のコプロセッサ・アクセスは、MMU命令に対応しており、コプロセッサ・インタフェースの結果として、MC68030が生成するものではありません。これらのサイクルは、MOVES命令によってのみ生成されます。詳細については、「第10章　コプロセッサ・インタフェースの説明」を参照してください。

7. 5　バス例外制御サイクル

MC68030のバス・アーキテクチャでは、バス・サイクルの完了を知らせる外部デバイスからの$\overline{DSACKx}$または$\overline{STERM}$をアサートする必要があります。$\overline{DSACKx}$、$\overline{STERM}$、または$\overline{AVEC}$は、次の場合にはアサートされません。

- 外部デバイスが応答しない。
- 割込みベクタがない。
- その他アプリケーションに依存する各種のエラーが発生した。

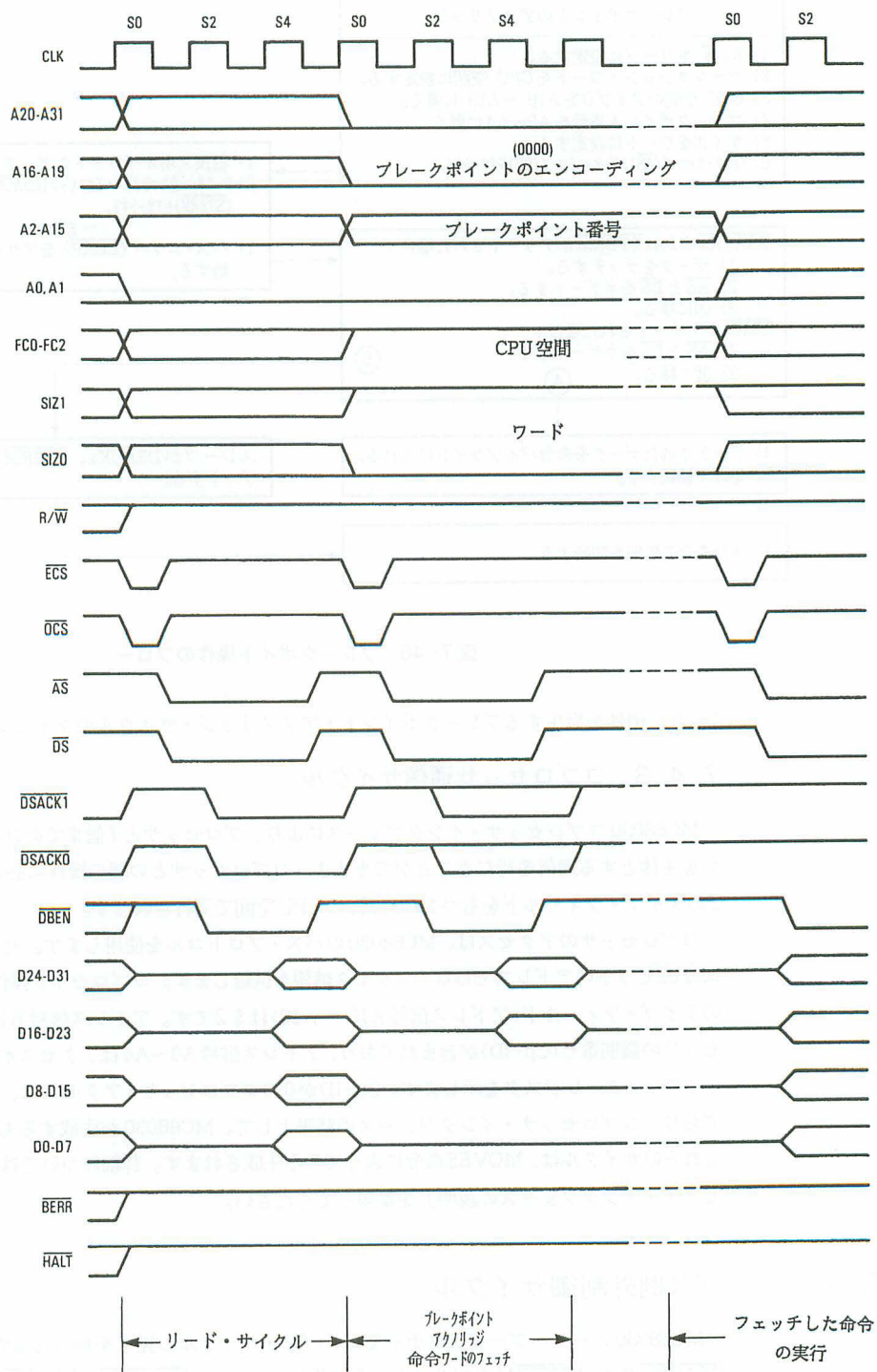



図7-47　ブレークポイント・アクノリッジ・サイクルのタイミング

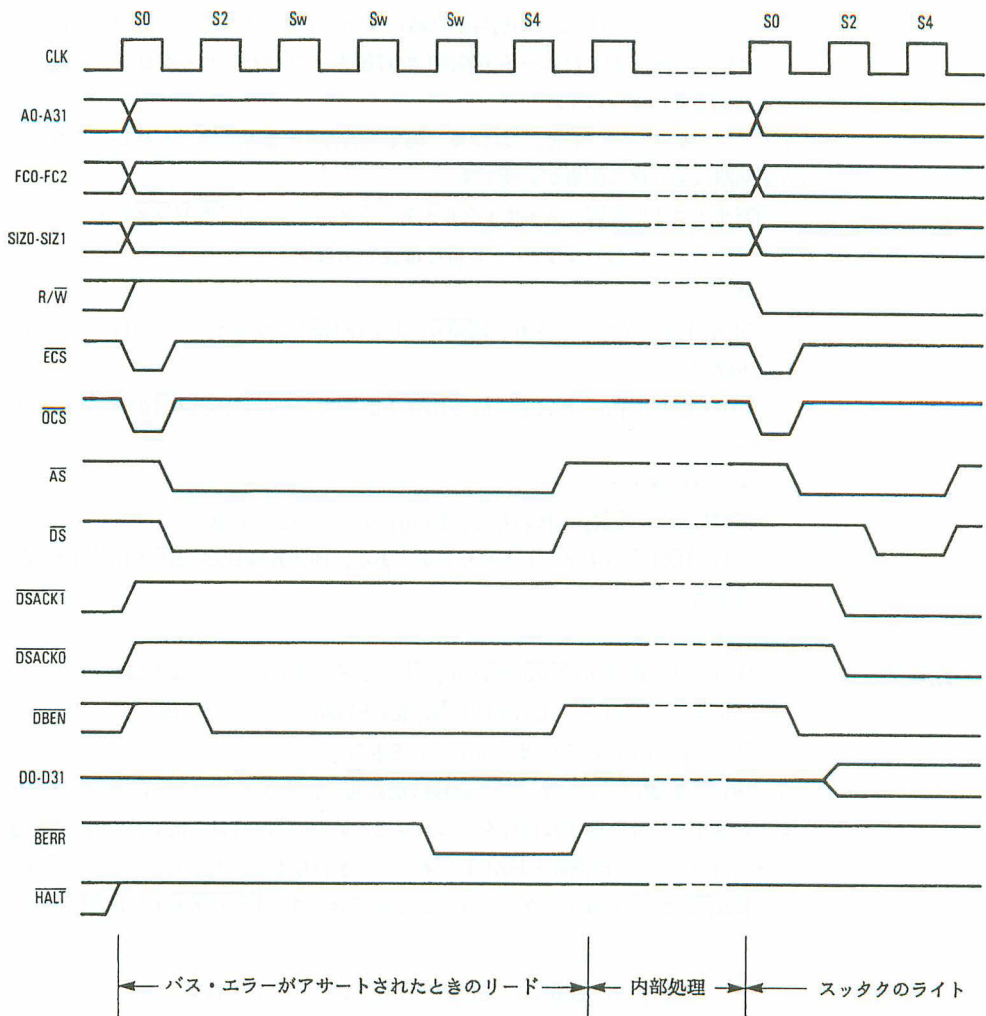



図7-48　ブレークポイント・アクノリッジ・サイクルのタイミング(例外発生の場合)

外部回路はプロセッサがアドレス・ストローブ($\overline{AS}$)をアサートした後、所定の時間内に$\overline{DSACKx}$、$\overline{STERM}$、または$\overline{AVEC}$をアサートして、これに応答するデバイスがないときには、バス・エラー信号($\overline{BERR}$)を発生させることができます。これにより、当該サイクルを終了し、プロセッサはそのエラー状態の例外処理に入ることができます。

メモリ管理ユニット(MMU)は内部エラーも検出することができます。これは、プロセッサがメモリで保護された領域のアドレスにアクセスしようとしたとき(たとえば、ユーザ・プログラムがスーパバイザ・データにアクセスしようとしたとき)、またはMMUがアドレス変換ディスクリプションのためのアドレス・テーブルをサーチしているときに、バス・エラーを受け取ると発生します。

バス例外制御に使用するもう1つの信号がホルト信号($\overline{HALT}$)です。この信号はデバッギングのために、外部デバイスがアサートして、シングル・バス・サイクル操作や($\overline{BERR}$と組み合わせて)エラーとなったバス・サイクルの再試行を行なうことができます。

再試行またはバス・エラーのときに、正しくバス・サイクルを終了させるために、$\overline{DSACKx}$、$\overline{BERR}$、および$\overline{HALT}$をMC68030の立上りエッジでアサートおよびネゲートするようにしておくとよいで

しょう。こうしておけば、2つの信号が同時にアサートされたときに、その両方に要求されるセットアップ時間(#47A)およびホールド時間(#47B)が、プロセッサ・クロックの同じ立下りエッジで満足されることになります。

タイミング条件については、「第13章　電気的特性」を参照してください。これらの信号を供給する外部回路にも同様な配慮が必要です。

非同期サイクルの許容バス・サイクル・ターミネーションとDSACKxのアサートとの関係をまとめると、次のようになります(ケース番号は表7-8参照)。

通常終了：

DSACKxがアサートされ、BERR、およびHALTがネゲートされたままである(ケース1)

ホルト終了：

DSACKxと同時またはその前にHALTがアサートされ、BERRがネゲートされたままである(ケース2)。

バス・エラー終了：

BERRがDSACKxの代わりに、その前(ケース3)または後(ケース4)、あるいは同時にアサートされ、HALTがネゲートされたままである。BERRはDSACKxと同時またはその後でネゲートされる。

再試行終了：

HALTとBERRがDSACKxの代わりに、その前(ケース5)または後(ケース6)、あるいは同時にアサートされる。BERRはDSACKxと同時、またはその後でネゲートされる。HALTはBERRと同時またはその後でネゲートできる。

表7-8に、制御信号シーケンスの各種の組合せと、それによってどのようにバス・サイクルが終了するかを示します。確実な操作を行なわせるために、「第13章　電気的特性」に記載する仕様の#28と#57に従って、BERRとHALTのネゲートを行なわなければなりません。DSACKx、BERR、およびHALTはASの後でネゲートすることができます。DSACKxまたはBERRが、次のバス・サ

表7-8　DSACK、BERR、HALTのアサートの結果

ケース番号	制御信号	ステートの立上りエッジでのアサート		結果
		N	N+2	
1	DSACKx BERR HALT	A NA NA	S NA X	通常サイクルを終了して動作を継続する。
2	DSACKx BERR HALT	A NA A/S	S NA S	通常サイクルを終了してホルトする。HALTがネゲートされていたときは動作を継続する。
3	DSACKx BERR HALT	NA/A A NA	X S NA	終了し、バス・エラー例外の処理を行なう。この処理は延期可能である。
4	DSACKx BERR HALT	A NA NA	X A NA	終了し、バス・エラー例外の処理を行なう。この処理は延期可能である。
5	DSACKx BERR HALT	NA/A A A/S	X S S	終了し、HALTがネゲートされていたら再試行する。
6	DSACKx BERR HALT	A NA NA	X A A	終了し、HALTがネゲートされていたら再試行する。

記号：N　－現在の偶数バス・ステート番号(例：S2, S4)。
A　－信号はこのバス・ステートでアサートされる。
NA－信号はこのバス・ステートではアサートされない。
X　－Don't Care
S　－信号が前のステートでアサートされ、このステートでもアサートされたままである。

表7-9　STERM、BERR、HALTのアサートの結果

ケース番号	制御信号	ステートの立上りエッジでのアサート N	N＋2	結果
1	STERM BERR HALT	A NA NA	－ － －	通常サイクルを終了して動作を継続する。
2	STERM BERR HALT	NA NA A/S	A NA S	通常サイクルを終了してホルトする。HALTがネゲートされていたときは動作を継続する。
3	STERM BERR HALT	NA A/S NA	A S NA	終了し、バス・エラー例外の処理を行なう。この処理は延期可能である。
4	STERM BERR HALT	A A NA	－ － －	終了し、バス・エラー例外の処理を行なう。この処理は延期可能である。
5	STERM BERR HALT	NA A A/S	A S S	終了し、HALTがネゲートされていたら再試行する。
6	STERM BERR HALT	A A A	－ － －	終了し、HALTがネゲートされていたら再試行する。

記号：N　－現在の偶数バス・ステート番号(例：S2, S4)。
A　－信号はこのバス・ステートでアサートされる。
NA －信号はこのバス・ステートではアサートされない。
X　－Don't Care
S　－信号が前のステートでアサートされ、このステートでもアサートされたままである。
－　－ステートN＋2はバス・サイクルの一部ではない。

イクルのS2までアサートされたままだと、そのサイクルが早く終了することがあります。

同期サイクルの終了信号は同期ターミネーション信号(STERM)です。STERMのアサートに関連して、同様にバス・サイクルを終了させる類似ケースが存在します。ただし、STERMとDSACKxの両方を同時に同じサイクルでアサートしてはなりません。STERMにはASがアサートされている間、プロセッサ・クロックの各立上りエッジに関して、セットアップ時間(＃60)とホールド時間(＃61)の要求条件が規定されています。バースト・サイクル中のバス・エラーおよび再試行終了は、「6. 1. 3. 2　バースト・モードの充てん」、「7. 5. 1　バス・エラー」、および「7. 5. 2　再試行操作」で述べるとおり動作します。

STERMでは、バス・サイクルの終了を、次のように要約することができます(ケース番号については表7-9を参照のこと)。

通常終了：

STERMがアサートされ、BERR、およびHALTがネゲートされたままである(ケース1)

ホルト終了：

STERMの前にHALTがアサートされ、BERRがネゲートされたままである(ケース2)。

バス・エラー終了：

BERRが、STERMの代わり、STERMと同時、またはその前(ケース3)、あるいはその後(ケース4)にアサートされ、HALTがネゲートされたままである。BERRがSTERMと同時またはその前にネゲートされる。

再試行終了：

HALTとBERRが、STERMの代わりに、同時に、またはその前(ケース5)、あるいはその後(ケース6)にアサートされる。STERMと同時にまたはその後にBERRがネゲートされる。HALTはBERRと同時またはその後にネゲートしてもよい。

例A：

システムはウォッチドッグ・タイマを使用して、未実装アドレス空間へのアクセスを終了させることができます。タイマはタイムアウト後、BERRをアサートします(ケース3)。

例B：

システムがRAM内容のエラー検出および修正を行ないます。システム設計者は次のことを行なうことができます。

1. データを検証するまでDSACKxを遅らせ、エラーを検出した場合は、BERRとHALTを同時にアサートして、プロセッサにそのエラー・サイクルを自動的に再試行するよう指示する(ケース5)か、データが有効の場合はDSACKxをアサートする(ケース1)。
2. データを検証するまでDSACKxを遅らせ、データにエラーがある場合はDSACKxと一緒に、あるいは単独でBERRをアサートする。これによって、この状態をソフトウェアで処理するための例外処理を開始する。
3. データを検証する前にDSACKxを返す。データが無効の場合は、次のクロック・サイクルでBERRをアサートする(ケース4)。これによって、この状態をソフトウェアで処理するための例外処理を開始する。
4. データを検証する前にDSACKxを返す。データが無効の場合は、次のクロック・サイクルでBERRとHALTをアサートする(ケース6)。これにより、メモリ・コントローラは自動再試行の前または実行中にRAMを修正することができる。

7.5.1 バス・エラー

バス・エラー信号を使用して、バス・サイクルおよび実行中の命令をアボートすることができます。BERRは「第13章　電気的特性」に記載されるタイミング条件に適合する場合は、DSACKxまたはSTERMに優先します。BERRがこれらの制限事項に適合しない場合は、MC68030が予測できない動作を行なう可能性があります。BERRが次のバス・サイクルまで、アサートされたままになっていると、そのサイクルの動作が正常に行なわれないことがあります。

バス・エラー信号が発生してバス・サイクルが終了すると、MC68030はそのバス・サイクルの直後から例外処理に入るか、その例外処理を延期することができます。命令プリフェッチ・メカニズムは、命令ワードが実行できる状態になる前に、バス・コントローラおよび命令キャッシュに命令ワードを要求します。命令フェッチでバス・エラーが発生した場合、プロセッサはその命令ワードを実際に使用するときがくるまで、そのバス・エラーの例外処理は行ないません。その前にある命令が分岐したり、タスク・スイッチが発生した場合は、バス・エラー例外は発生しません。

バス・エラー信号は、次のいずれかの場合のバス・サイクルにおいて認識されます。

- DSACKx(またはSTERM)およびHALTがネゲートされ、BERRがアサートされている。
- HALTおよびBERRがネゲートされ、DSACKxがアサートされている。BERRはその後1クロック・サイクル以内にアサートされる(HALTはネゲートされたままである)。
- BERRがアサートされ、STERMがアサートされ認識された立上りクロック・エッジの次の立下りエッジで認識される(HALTはネゲートされたままである)。

プロセッサがバス・エラー状態を認識すると、通常どおりに現在のバス・サイクルを終了します。図7-49にDSACKxとSTERMのどちらもアサートされない場合のバス・エラーのタイミングを示します。図7-50にDSACKxの後でアサートされるバス・エラーのタイミングを示します。いずれの場合も例外処理が行なわれます(バス・エラー例外処理の詳細については、「8.1.2　バス・エラー例外」を参照してください)。いずれかのオンチップ・キャッシュにデータを供給するリード・サイクルでBERRがアサートされると、キャッシュのデータは無効としてマークされます。しかし、データ・キャッシュにデータを書き込むライト・サイクルが外部バス・エラーになっても、キャッシュ内のデータは無効としてマークされません。

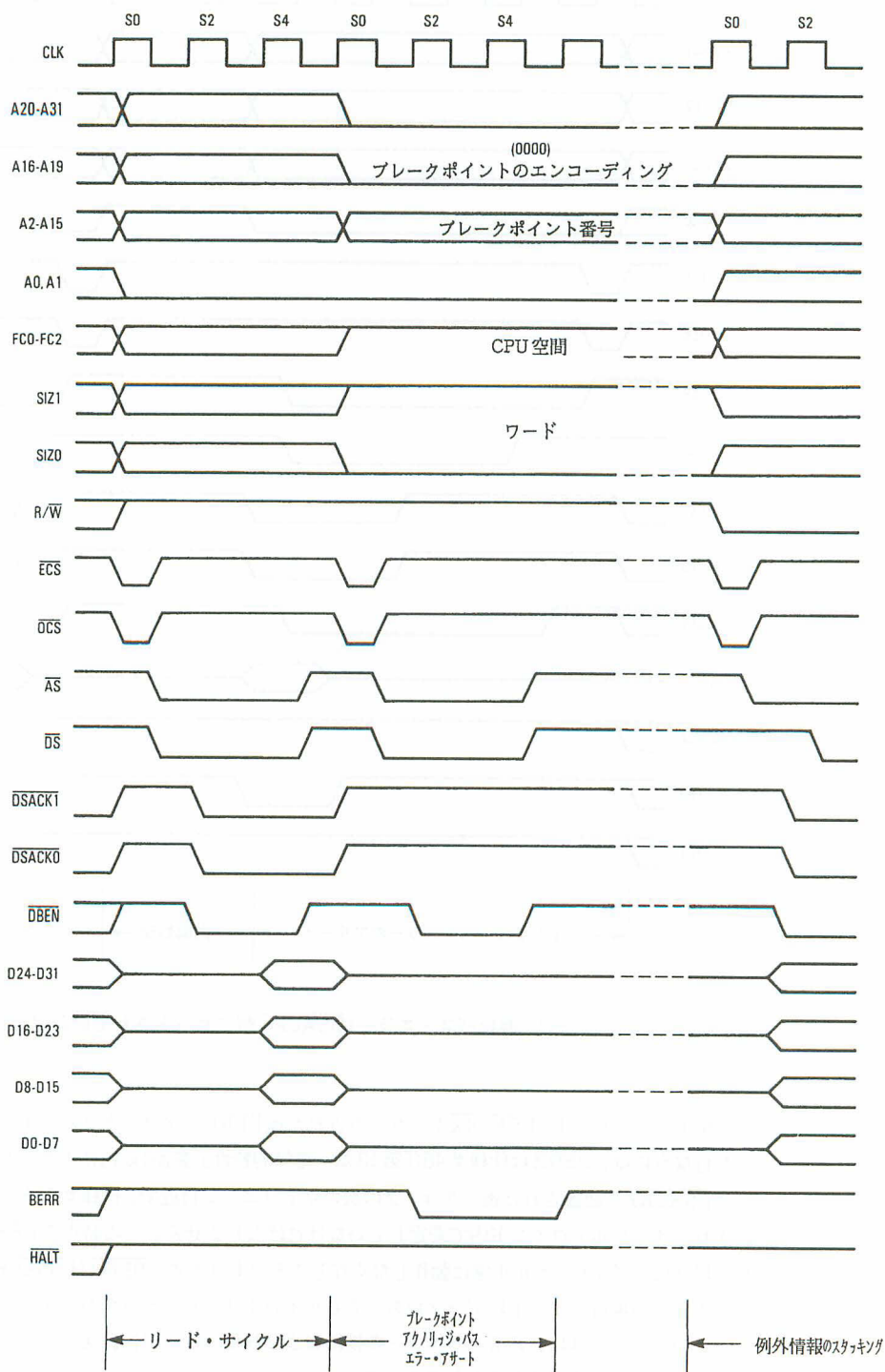



図7-49　バス・エラーのタイミング（DSACKxがネゲートされている場合）

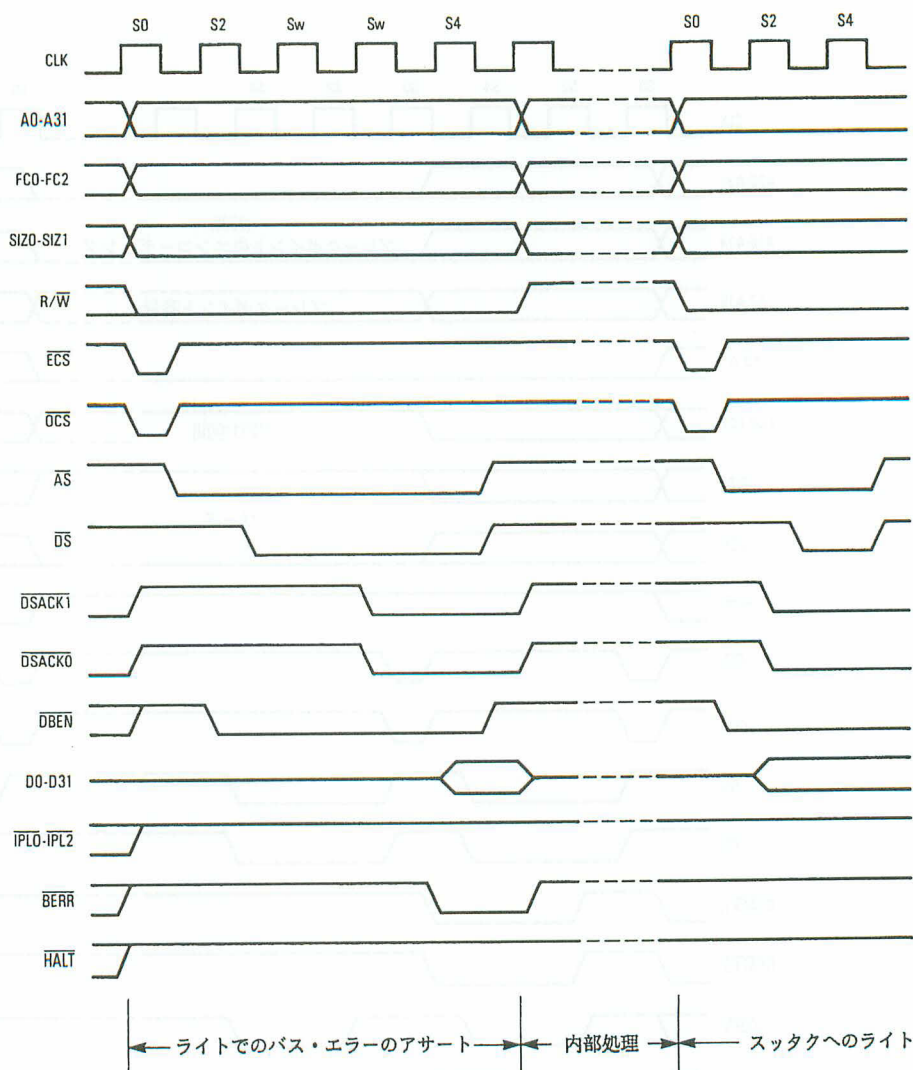



図7-50　遅延バス・エラー(DSACKxがアサートされている場合)

2番目のケースでは、DSACKxがアサートされた後BERRがアサートされますが、完全な同期動作を行なうには、BERRは仕様#48(「第13章　電気的特性」参照)以内にアサートされるか、あるいはDSACKxが認識された後、クロックの次の立下りエッジ付近で、仕様#27Aおよび#47Bで定義されるサンプル・ウインド内で安定していなければなりません。この時点でBERRが安定していない場合は、プロセッサが正常に動作しなくなることがあります。BERRはDSACKxよりも優先されます。この場合、バス上にデータがあってもかまいませんが、それが有効かどうかはわかりません。このシーケンスは、メモリ・エラーの検出および修正ロジック回路をもつシステムや外部キャッシュ・メモリが使用することができます。

3番目のケースで述べるBERRのアサート(STERMの後で認識される)は、その前のパラグラフで述べた条件とよく似ています。BERRは、仕様#27Aおよび#28Aで定義するように、クロックの次の立下りエッジに対するサンプル・ウインド内で、安定していなければなりません。図7-51にこの場合のタイミングを示します。

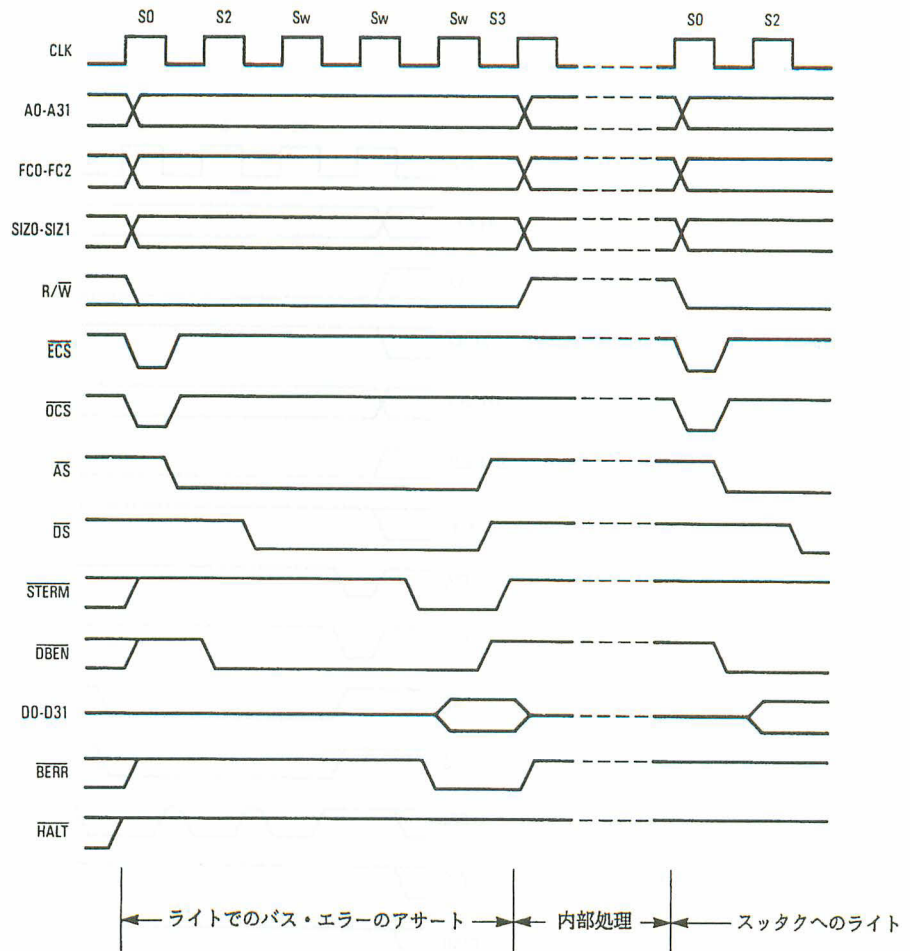



図7-51 $\overline{\text{STERM}}$がアサートされている場合の遅延バス・エラー――例外処理を実行

バースト充てん操作中に発生するバス・エラーは特殊なケースです。バーストの最初のサイクル中にバス・エラーが発生した場合は、データは無視され、キャッシュ・ライン全体が無効としてマークされ、そのバースト操作はアボートされます。このサイクルが命令キャッシュの場合、バス・エラー例外が保留されます。このバス・エラーは、実行ユニットがバス・サイクル中にラッチした2ワードのいずれかを使用しようとしたときにだけ処理されます。このサイクルがデータ・フェッチの場合は、直ちにバス・エラー例外の処理に入ります。パイプライン操作の詳細については、「第11章 命令実行時間」を参照してください。

バースト・モードに入った後にバス・エラーが発生した(つまり、2回目以降のアクセスで)ときは、プロセッサはバースト操作を終了し、そのサイクルに対応するキャッシュ・エントリは無効としてマークされますが、図7-52に示すとおり、プロセッサは例外処理を行ないません。第2サイクルがミスアラインメントのオペランドのフェッチの場合、プロセッサは図7-53に示すとおり、$\overline{\text{CBREQ}}$をネゲートして2番目の部分に対する別のリード・サイクルを実行します。再度$\overline{\text{BERR}}$がアサートされた場合は、MC68030は例外処理を行ないます。バースト充てん操作中、MC68030は遅延バス・エラーをサポートします。このタイミングは、通常の同期サイクルでの遅延バス・エラーに対する$\overline{\text{STERM}}$およびクロックと同じです。

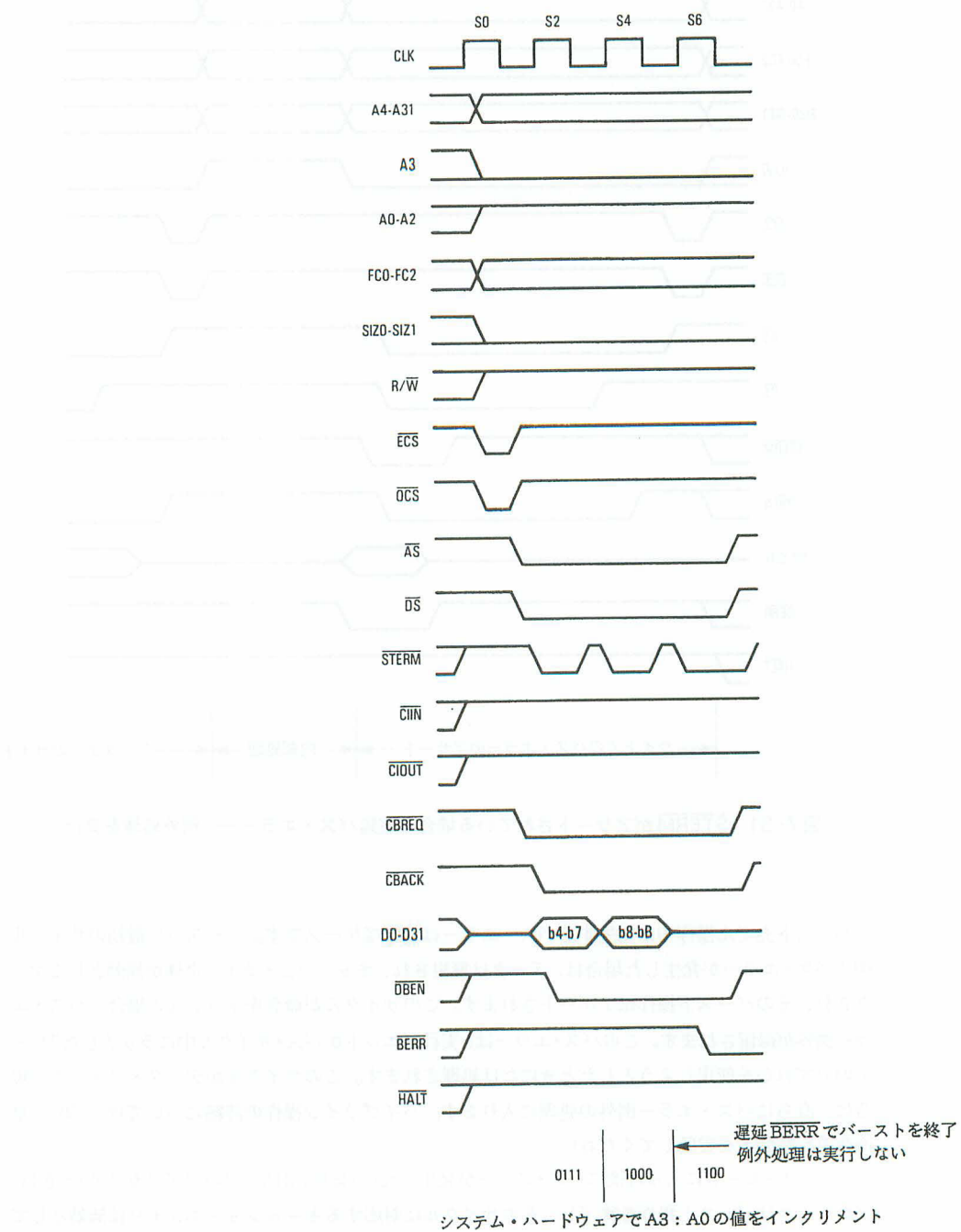



図7-52　ロング・ワード・オペランドの要求——3回目のアクセスまでBERRを遅延

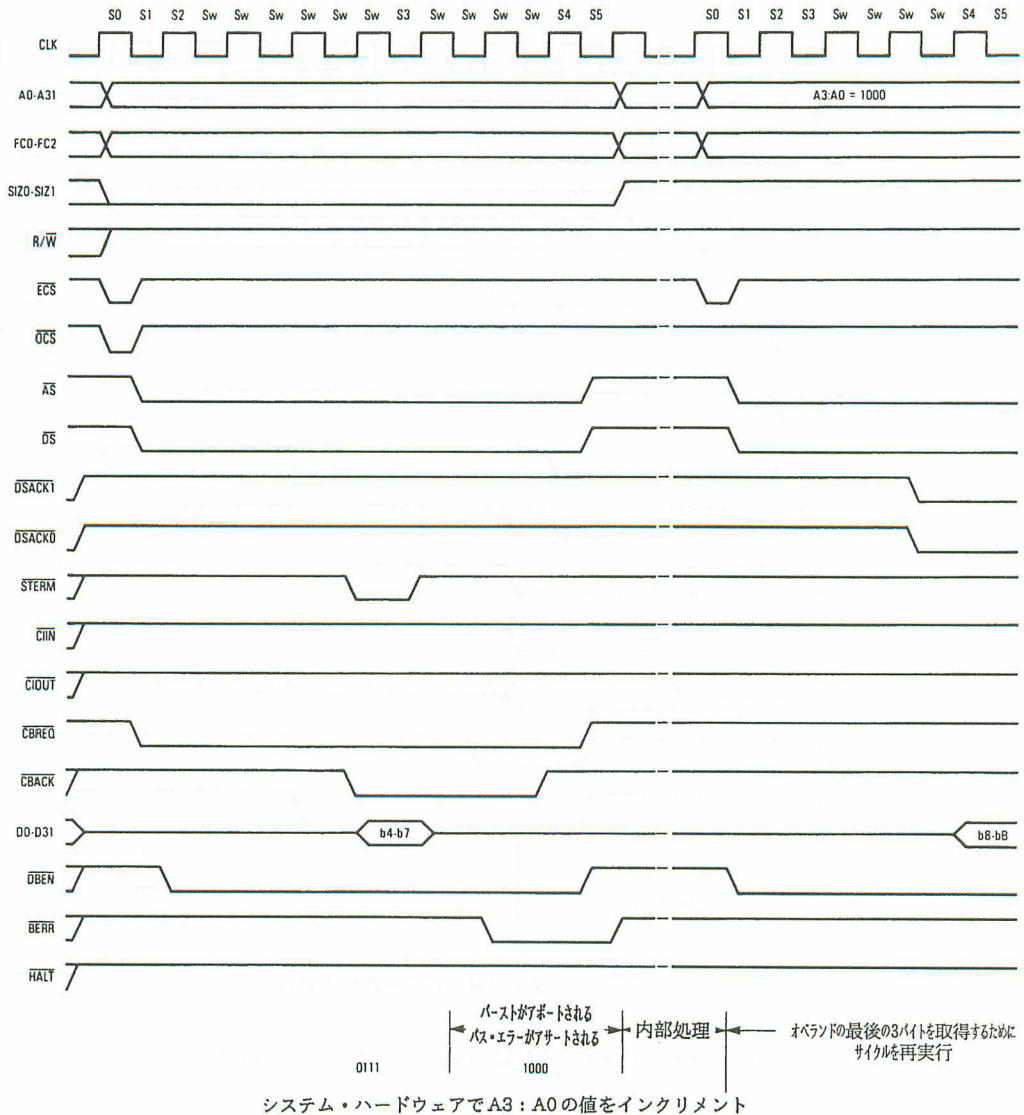



図7-53　ロング・ワード・オペランドの要求――2回目のアクセスでBERR

7.5.2　再試行操作

バス・サイクルの間に、BERR信号とHALT信号の両方が外部デバイスによってアサートされると、プロセッサは再試行シーケンスに入ります。この場合も前出の遅延バス・エラー信号の場合と同様に、同期および非同期サイクルの両方に対して再試行を遅延させることができます。

プロセッサはそのバス・サイクルを終了した後、制御信号を非アクティブ状態にし、HALT信号が外部回路によってネゲートされるまで次のバス・サイクルを実行しません。非同期遅延の後、プロセッサは同じアクセス情報(アドレス、ファンクション・コード、サイズなど)を使用して前のサイクルを再試行します。再試行サイクルを正しく実行するために、リード・サイクルのS2の前にBERR信号をネゲートしなければなりません。図7-54に非同期サイクルの再試行操作を図7-55に同期サイクルの再試行操作を示します。

プロセッサは、どのリード・モディファイ・ライト操作のリードまたはライト・サイクルでも、別々

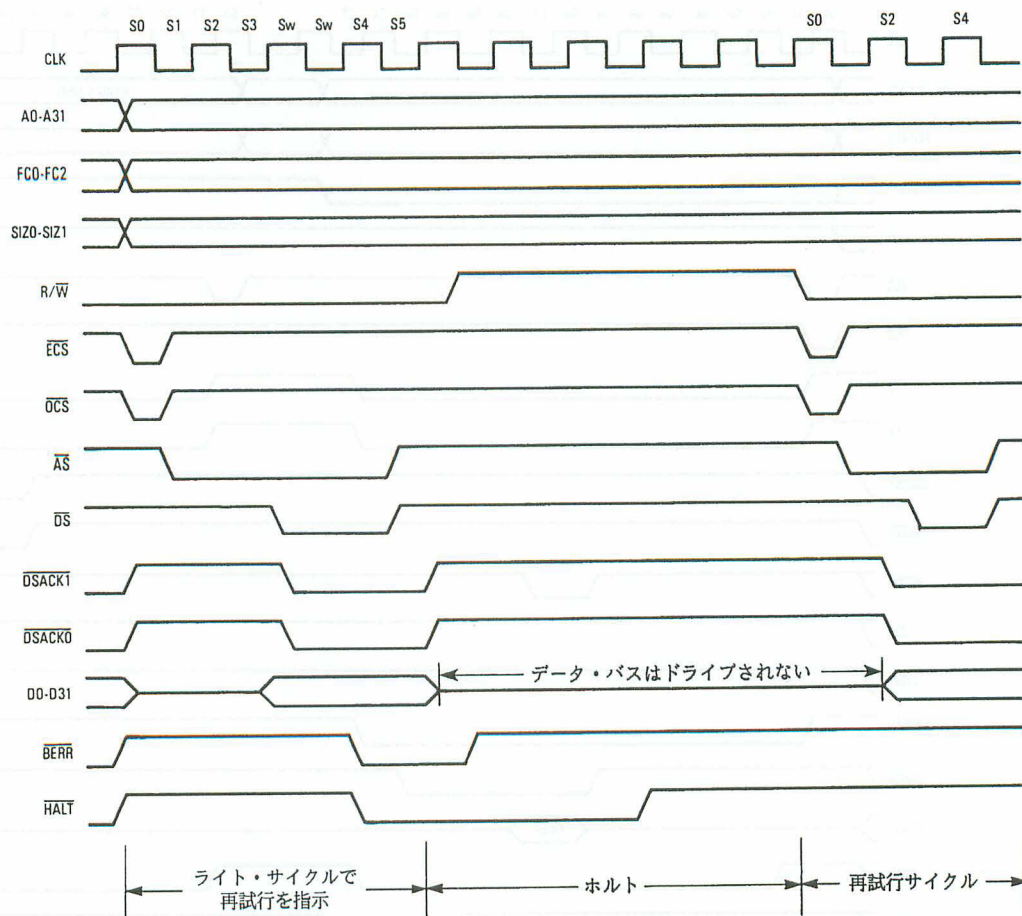



図7-54　非同期遅延再試行

に再試行することができます。これは、再試行シーケンスの全期間で、$\overline{\mathrm{RMC}}$信号がアサートされたままになっているためです。

バースト操作の初期アクセスで、再試行($\overline{\mathrm{BERR}}$および$\overline{\mathrm{HALT}}$をアサートして示す)が指示されると、プロセッサはバス・サイクルを再試行し、再び$\overline{\mathrm{CBREQ}}$をアサートします。図7-56に初期バースト操作を繰り返し実行する遅延再試行操作を示します。しかし、バースト操作の第2、第3、または第4サイクル中に$\overline{\mathrm{BERR}}$と$\overline{\mathrm{HALT}}$を同時にアサートして再試行の通知を行なった場合、要求されたオペランドがミスアラインメントであっても、再試行操作は行ないません。バースト操作の後続サイクルで$\overline{\mathrm{BERR}}$および$\overline{\mathrm{HALT}}$をアサートすると、$\overline{\mathrm{BERR}}$と$\overline{\mathrm{HALT}}$操作が別々に行なわれます。外部バスの動作は、$\overline{\mathrm{HALT}}$がネゲートされ、プロセッサはバースト操作中のバス・エラーのところで説明したとおり動作します。

$\overline{\mathrm{BERR}}$および$\overline{\mathrm{HALT}}$とともにバス要求信号($\overline{\mathrm{BR}}$)をアサートすると、バスの放棄と再試行操作を行ないます。MC68030はリード・モディファイ・ライト操作中には、バスを放棄しません。リード・モディファイ・ライト・サイクル中に、プロセッサに対して、バスを放棄してバス・サイクルの再試行を行なうように要求するデバイスは、必ず$\overline{\mathrm{BERR}}$と$\overline{\mathrm{BR}}$をアサートしなければなりません($\overline{\mathrm{HALT}}$はアサートしてはなりません)。バス・エラー処理ソフトウェアは、特殊ステータス・ワード(「8. 2. 1　特殊ステータス・ワード」参照)のRMビットを調べ、それがセットされていた場合はこのフォールトを解決するための適切な処置を取る必要があります。

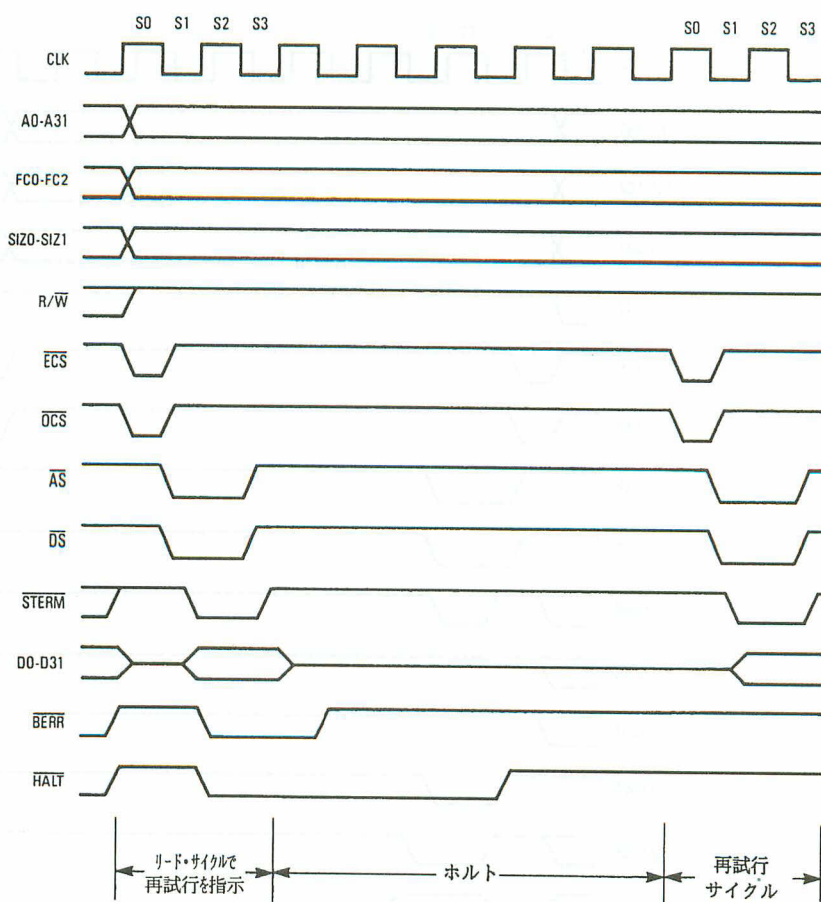



図7-55　同期遅延再試行

7.5.3　ホルト操作

ホルト信号($\overline{\text{HALT}}$)がアサートされ、$\overline{\text{BERR}}$がアサートされていないとき、MC68030は次のバス・サイクルで外部バス動作を停止します。$\overline{\text{HALT}}$だけではバス・サイクルを終了しません。正しいタイミング条件に従って$\overline{\text{HALT}}$をネゲートし再アサートすると、シングル・ステップ(バス・サイクル単位)操作を行なうことができます。$\overline{\text{HALT}}$信号は外部バス・サイクルにしか影響を与えませんので、命令キャッシュの中に存在し、データの書込み(あるいはデータ・キャッシュでミスを起こす読出し)を行なわないプログラムは、$\overline{\text{HALT}}$信号の影響を受けず、実行を継続することができます。

シングル・サイクル・モードでは、ユーザは外部プロセッサ操作を1バス・サイクルずつ進める(そして、デバッグする)ことができます。図7-57にシングル・サイクル操作の必要条件を示します。$\overline{\text{HALT}}$がアサートされているときに、バス・エラーが発生すると再試行操作が行なわれますので、ユーザはシングル・サイクル・モードでデバッギングを行なっている間に、再試行サイクルの発生を予測できます。このシングル・ステップ操作とソフトウェア・トレース機能により、システム・デバッガは単一バス・サイクル、単一命令、またはプログラム・フローの変化をトレースすることができます。これらのプロセッサの機能とソフトウェア・デバッギング・パッケージを組み合わせれば、完全なデバッギング機能が実現可能です。

プロセッサが$\overline{\text{HALT}}$信号をアサートしてバス・サイクルを完了すると、データ・バス(D0-D31)

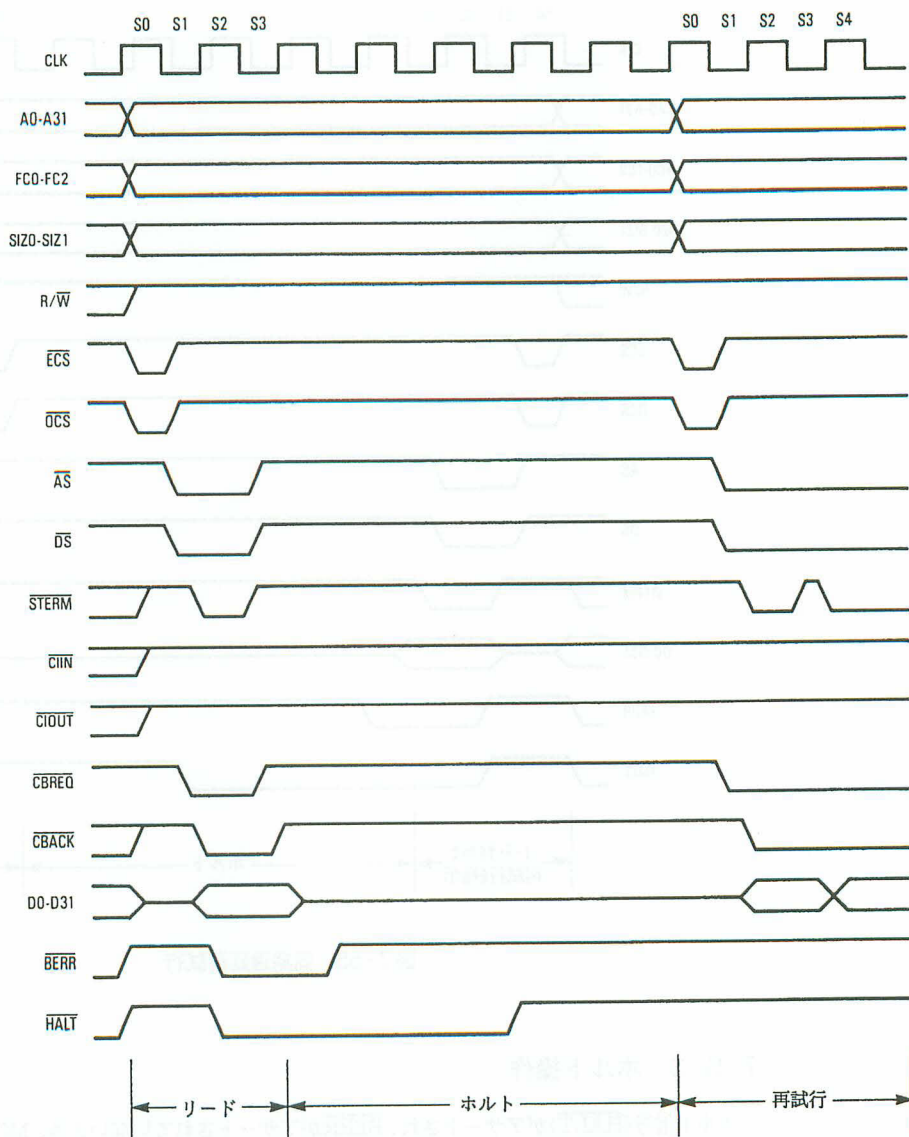



図7-56　バーストのための遅延再試行操作

がハイ・インピーダンス状態に置かれ、バス制御信号が非アクティブ状態(ハイ・インピーダンス状態ではない)にドライブされますが、アドレス、ファンクション・コード、サイズ、およびリード/ライトの各信号は同じ状態のままです。ホルト操作は、バス調停には影響を与えません(「7.7　バス調停」参照)。MC68030がホルト状態のときにバス調停が発生すると、アドレスと制御信号もハイ・インピーダンス状態に置かれます。バスの制御権がMC68030に返ったときに、まだ$\overline{\mathrm{HALT}}$がアサートされたままであれば、アドレス、ファンクション・コード、サイズ、およびリード/ライトの各信号が再び以前の状態にドライブされます。プロセッサはホルト状態のときには、割込み要求のサービスは行ないませんが、必要に応じて$\overline{\mathrm{IPEND}}$信号をアサートすることは可能です。

7.5.4　ダブル・バス・フォールト

以前のバス・エラー、アドレス・エラー、またはリセット例外に対する例外処理シーケンス中に、

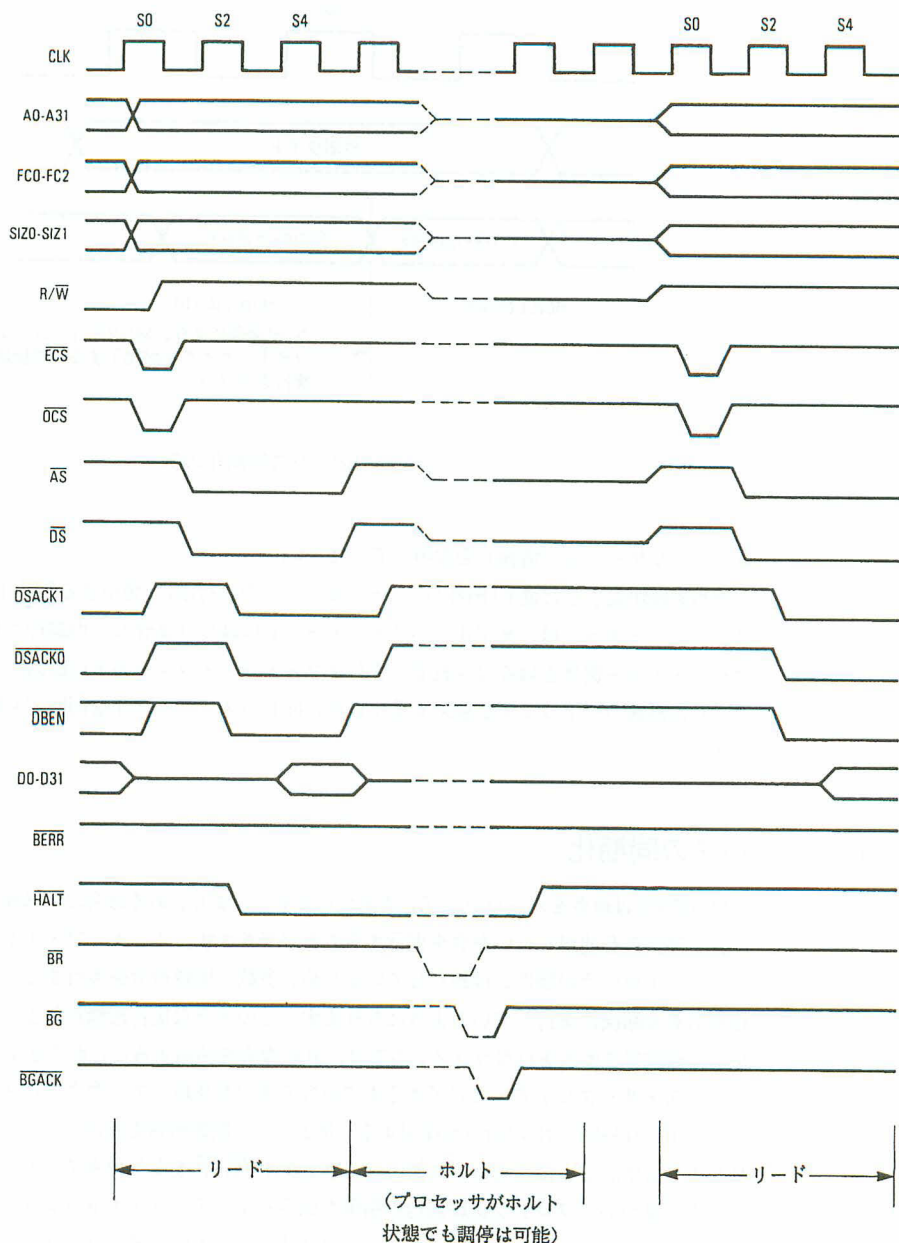



図7-57　ホルト操作のタイミング

新たにバス・エラーまたはアドレス・エラーが発生すると、ダブル・バス・フォールトになります。

たとえば、プロセッサがバス・エラー例外を処理している間に、マシンの状態に関する情報を含むいくつかのワードをスタックしようとしたとします。このスタック操作中にバス・エラー例外が発生した場合、2回目のエラーはダブル・バス・フォールトとみなされます。停止したプロセッサを再スタートするには、外部リセットによる以外方法はありません。ただし、この場合にもバス調停は行なわれます(「7.7　バス調停」参照)。

MC68030は、リセットされるまでSTATUS信号を継続してアサートして、ダブル・バス・フォールト状態が発生したことを知らせます。STATUSを1、2、または3クロック期間だけアサートし、他のマイクロシーケンサのステータス表示を知らせます。STATUS信号の解釈については、「第12

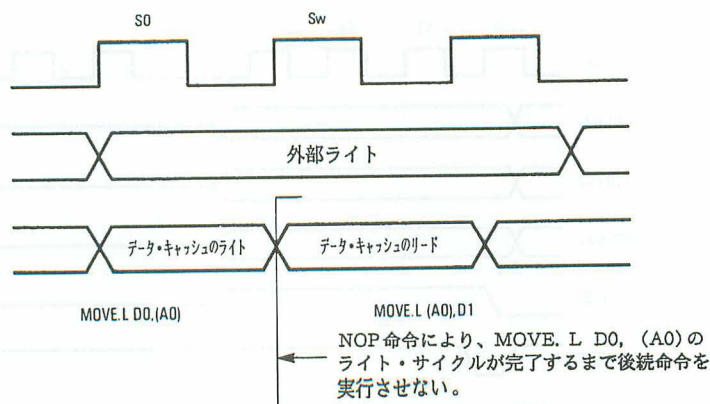



図7-58　バス同期化の例

章　アプリケーション情報」を参照してください。

例外処理が完了した後で(例外ハンドラ・ルーチンの実行中に)発生する2回目のバス・エラーまたはアドレス・エラーは、ダブル・バス・フォールトにはなりません。再試行されるバス・サイクルはバス・エラー例外とはみなされず、またダブル・バス・フォールトの原因にもなりません。プロセッサは外部ハードウェアが要求するかぎり、同じバス・サイクルを何回でも繰り返し実行し続けます。

7.6　バスの同期化

MC68030は命令をオーバラップして実行します。つまり、ある命令のバス操作を行なっている間に、外部バスを使用しない命令を実行することができます。オンチップ・キャッシュの操作は、バス・コントローラの操作とは独立しているため、多数の後続命令を実行することができ、見かけ上は命令を非順次に実行しているようになります。このような実行形態が望ましくなく、バス動作に従って順次アクセスを行なうシステムでは、NOP命令を活用することができます。NOP命令は保留中のバス・サイクルがすべて完了するまで命令の実行を凍結して、命令とバスを同期させます。

このような目的にNOP命令を使用する一例として、制御情報を外部レジスタに書き込むライト操作があります。この操作では、外部ハードウェアが$\overline{\text{BERR}}$を条件付きでアサートし、書き込まれるデータに基づいてプログラムの実行の制御を試みます。データ・キャッシュがイネーブルされ、ライト・サイクルでデータ・キャッシュのヒットが起こった場合、そのキャッシュが更新されます。そのデータは、外部ライト・サイクルが完了する前に、後続の命令で使用することができます。MC68030は現在のバス・サイクルが終わるまで、バス・エラーを処理することはできないため、外部ハードウェアがプログラムの実行の流れを変えることはできなかったはずです。外部サイクルが完了するまで後続の命令を実行しないようにするには、ライト操作を起動する命令の後にNOP命令を挿入しておきます。そうすれば、ライト操作の直後で、後続の命令が実行される前にバス・エラー例外処理に入ります。これはあくまでも例外的なケースであり、大部分のシステムではこのような目的にNOP命令を使用する必要はありません。

エラー検出/修正回路を備えたシステムでも、同期化にNOPは必要ありません。MMUはデータ・キャッシュの操作を行なう前に、常にライト・サイクルの妥当性をチェックしており、これらは外部で実行されるため、MC68030がキャッシュに正しいデータを書き込むことが保証されています。したがって、外部バス・エラーがエラーを知らせる前に、後続の命令がキャッシュから誤ったデー

タを取り出して使用する危険はありません。

バス同期化の例を図7-58に示します。

7.7　バス調停

MC68030のバスの設計では、任意の時点でプロセッサまたは外部デバイスのいずれか1つがバス・マスタになっています。バス上の1つまたは複数の外部デバイスがバス・マスタになることができます。バス調停とは、外部デバイスがそれを使用してバス・マスタになるためのプロトコルです。MC68030のバス・コントローラは、プロセッサの優先順位が最も低くなるようにバス調停信号を管理しています。バスを獲得する必要のある外部デバイスは、以下のパラグラフで述べるシーケンスによりバス調停信号をアサートしなければなりません。バス・マスタになりうるデバイスがいくつかあるシステムでは、それらのデバイスに優先順位を割り当てる外部回路が必要です。それによって、2つ以上の外部デバイスが同時にバス・マスタになろうとしたときには、最も高い優先順位をもつデバイスが最初にバス・マスタになります。このプロトコルのシーケンスは次のとおりです。

1. 外部デバイスがバス要求信号をアサートする。
2. プロセッサはバス許可信号をアサートし、現在のバス・サイクルの終わりでバスが使用可能になることを示す。
3. 外部デバイスはバス許可アクノリッジ信号をアサートして、自分がバスの制御権を得たことを示す。

バス・サイクル中またはサイクルとサイクルの間で、いつでもバス要求信号($\overline{\mathrm{BR}}$)を出すことができます。バス許可信号($\overline{\mathrm{BG}}$)は$\overline{\mathrm{BR}}$の応答としてアサートされます。バス許可信号($\overline{\mathrm{BG}}$)は、通常MC68030が内部でバス・サイクルを実行する決定を行なったときを除いて、$\overline{\mathrm{BR}}$が同期化され認識

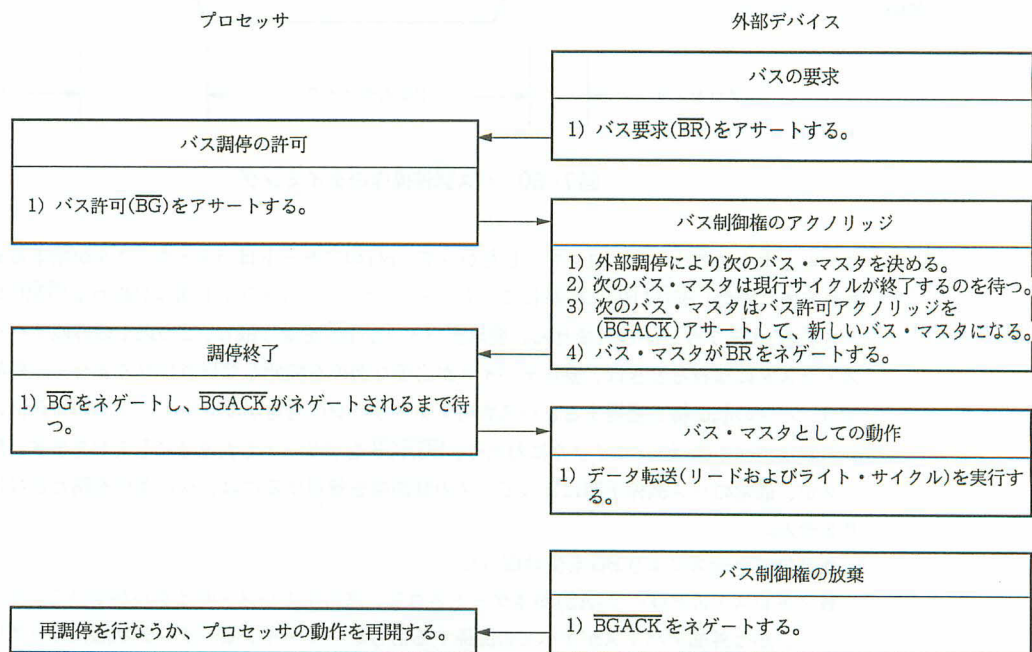



図7-59　単独バス要求に対するバス調停フローチャート

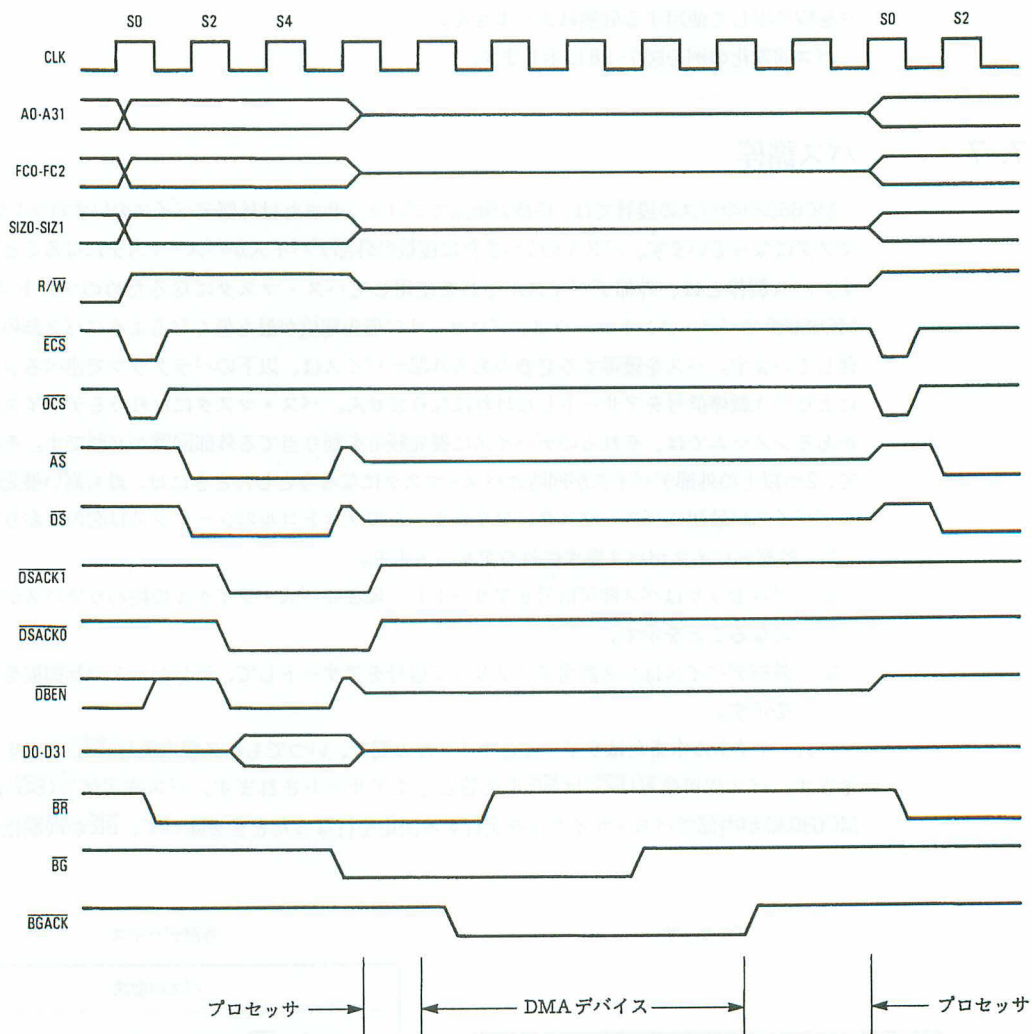



図7-60　バス調停操作のタイミング

されるとすぐにアサートされます。したがって、$\overline{BG}$のアサートはバス・サイクルが始まるまで延期されます。また、$\overline{BG}$は$\overline{BR}$に応答して、リード・モディファイ・ライト操作が終わる($\overline{RMC}$がネゲートされる)までアサートされません。要求デバイスが$\overline{BG}$を受け取り、2つ以上の外部デバイスがバス・マスタになれるときは、要求デバイスが必要な調停を開始しなければなりません。外部デバイスは、バスの制御権を獲得するとバス許可アクノリッジ信号をアサートし、バス・マスタになっている間はすべてのバス・サイクルにわたり、$\overline{BGACK}$をアサートしたままにしておきます。外部デバイスが、通常のバス調停手順によってバスの制御権を獲得するには、次の条件を満たさなければなりません。

- 調停プロセスにより$\overline{BG}$を受け取った。
- アドレス・ストローブ($\overline{AS}$)がネゲートされて、進行中のバス・サイクルがないことを示しており、かつ外部デバイスがすべての関係する信号がハイ・インピーダンス状態になっていることを保証しなければならない(「第13章　電気的特性」の仕様#7を守ることによって)。
- 直前のサイクルに対する終了信号($\overline{DSACKx}$または$\overline{STERM}$)が非アクティブになり、外部デバイスがバスから切り離されていることを示している(これはオプションであり、「7.7.3　バス

許可アクノリッジ」を参照のこと)。

● $\overline{\text{BGACK}}$が非アクティブになっており、ほかにバスの制御権を要求しているバス・マスタがないことを示している。

図7-59に単一デバイスのバス調停に関する詳細を示します。図7-60はこの操作のタイミング図です。この方法により、データ転送サイクルでのバス要求を処理することができます。

タイミング図では、$\overline{\text{BGACK}}$がアサートされた時点で$\overline{\text{BR}}$がネゲートされています。この種の操作は、プロセッサとバス・マスタになりうる1つのデバイスで構成されるシステムに適用されます。バス・マスタになりうるデバイスが複数あるシステムでは、各デバイスからのバス要求ラインをワイヤードORして、プロセッサに入力することができます。そのようなシステムでは、2つ以上のバス要求が同時にアサートされる可能性があります。

図7-60のタイミング図では、バス許可アクノリッジ信号が遷移してから数サイクル後に$\overline{\text{BG}}$がネゲートされています。しかし、バス許可信号がネゲートされた後まだバス要求が保留されている場合、プロセッサはバス許可がネゲートされてから数クロック・サイクル以内に、再度バス許可信号をアサートします。このバス許可信号によって、外部の調停回路は現在のバス・マスタがバスの使用を終える前に、次のバス・マスタを選択することができます。以下のパラグラフで、調停プロセスでの3つのステップについて詳しく説明します。

バス調停要求は、通常の処理、$\overline{\text{RESET}}$のアサート、$\overline{\text{HALT}}$のアサートで認識されるほか、ダブル・バス・フォールトのためにプロセッサがホルト状態になったときにも認識されます。

7. 7. 1 バス要求

バス・マスタになりうる外部デバイスは、バス要求($\overline{\text{BR}}$)信号をアサートしてバスを要求します。この信号はワイヤードOR接続が可能で(必ずしもオープン・コレクタ・デバイスでなくてよい)、プロセッサに外部デバイスのどれかがバスの制御権を要求していることを示します。プロセッサは実質的には外部デバイスより低いバス優先レベルにあり、現在のバス・サイクル(すでに開始している場合)を完了した後バスを放棄します。

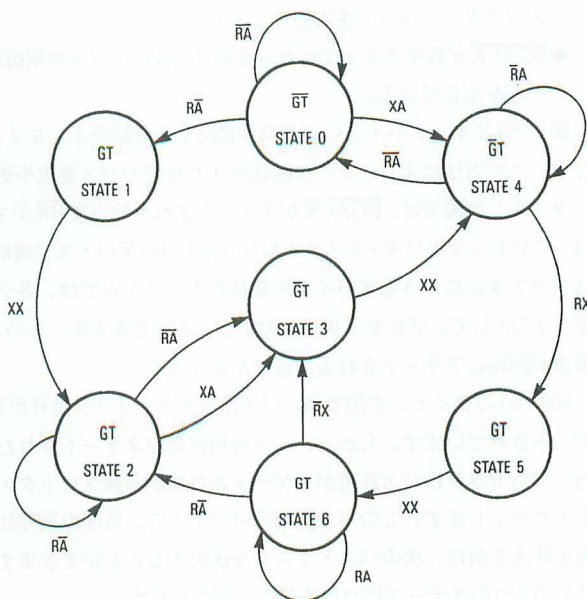
バス要求信号がアクティブになっている間に、アクノリッジ信号が受信されなかった場合、プロセッサはいったんバス要求をネゲートし、そのままバス・マスタを続けます。これにより、調停回路が誤ってノイズに応答したり、外部デバイスがバスの使用を許可される前にそれを使用する必要がなくなった場合にも、通常の処理に妨害を与えないようになっています。

7. 7. 2 バス許可

プロセッサはバス要求を受け取った後、できるだけ早くバス許可($\overline{\text{BG}}$)信号をアサートします。このアサートは、リード・モディファイ・ライト・サイクル中を除いては内部で同期された直後、またはバス・サイクルを実行する内部決定の直後に行なわれます。リード・モディファイ・ライト・サイクル中には、プロセッサはその操作がすべて完了するまでは、バス許可信号をアサートしません。リード・モディファイ・ライト・サイクルでは、リード・モディファイ・ライト・サイクル($\overline{\text{RMC}}$)信号がアサートされ、バスがロックされていることを示します。別のバス・サイクルを実行する内部決定を行なった場合、$\overline{\text{BG}}$はバス・サイクルが開始されるまで延期されます。

バス許可信号は、ディジー・チェイン回路または特定の優先順位エンコード回路をとおして伝達することができます。プロセッサは、プロトコルに従うかぎりどのような種類の外部調停でも受け入れます。

7. 7. 3 バス許可アクノリッジ

バス許可信号を受け取ると、バス要求を出しているデバイスはアドレス・ストローブ($\overline{\text{AS}}$)、デー

R－バス要求
A－バス許可アクノリッジ
G－バス許可
T－バス制御回路へのスリー・ステート制御
X－Don't Care

注：$\overline{\text{RMC}}$がアサートされている間$\overline{\text{BG}}$出力はアサートされない。

図7-61　バス調停の状態図

タ転送、およびサイズ・アクノリッジ($\overline{\text{DSACKx}}$)(または同期ターミネーション、$\overline{\text{STERM}}$)、およびバス許可アクノリッジ($\overline{\text{BGACK}}$)がネゲートされるまで待ってから、自分自身の$\overline{\text{BGACK}}$をアサートします。$\overline{\text{AS}}$のネゲートは、前のバス・マスタが仕様#7(「第13章　電気的特性」参照)のあと、そのバスを解放することを示します。$\overline{\text{DSACKx}}$または$\overline{\text{STERM}}$のネゲートは、前のスレーブ・デバイスが前のバス・マスタとのサイクルを終了したことを意味します。アプリケーションによっては、$\overline{\text{DSACKx}}$をこのような方法で使用できないこともあります。

その場合は、$\overline{\text{AS}}$だけに依存するように汎用デバイスが接続されます。$\overline{\text{BGACK}}$がアサートされると、そのデバイスは自分で$\overline{\text{BGACK}}$をネゲートするまでバス・マスタになっています。$\overline{\text{BGACK}}$は代替バス・マスタで要求されるすべてのバス・サイクルが完了するまでネゲートしてはなりません。バスの制御権は$\overline{\text{BGACK}}$がネゲートされた時点で消滅します。バスの使用を許可されたデバイスからのバス要求は、$\overline{\text{BGACK}}$がアサートされたあと、ネゲートしなければなりません。$\overline{\text{BGACK}}$がアサートされたあともまだバス要求が保留になっている場合は、バス要求がネゲートされてから数クロック以内に別のバス許可がアサートされます。詳細は「7.7.4　バス調停制御」を参照してください。この場合、プロセッサは$\overline{\text{BG}}$を再アサートしてからでないと、外部バス・サイクルを実行しません。

7.7.4　バス調停制御

MC68030のバス調停制御ユニットは、有限のステート・マシンで実現されています。前述したとおり、MC68030のすべての非同期入力信号は、プロセッサ・クロック最大2サイクル以内に内部で同期化されます。

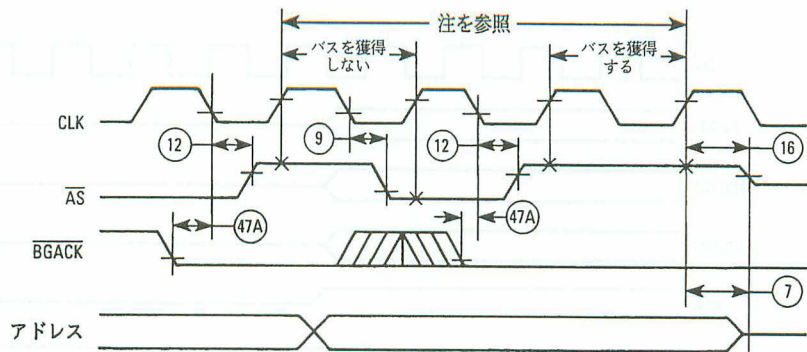
図7-61に示すように、RおよびAという名前の入力信号は、それぞれバス要求信号とバス許可ア

注：代替バス・マスタは、連続したクロックの2回目の立上りエッジで$\overline{\text{AS}}$“H”をサンプルした後($\overline{\text{BGACK}}$の“L”が認識された後)でバスを獲得しなければなりません。

図7-62　シングル・ライト・バス調停のタイミング図

クノリッジ信号を内部的に同期化したものです。バス許可出力はG、そして内部スリー・ステート制御信号はTになっています。Tが「真」の場合、アドレス、データ、および制御の各バスは、アドレス・ストローブ($\overline{\text{AS}}$)およびリード・モディファイ・ライト($\overline{\text{RMC}}$)信号のネゲートに続く立上りエッジの後、ハイ・インピーダンス状態に置かれます。すべての信号は、実際のアクティブ電圧レベルには関係なく正論理(アクティブ“H”)で示されています。

ステートの変化は、内部信号が有効になったあと、クロックの次の立上がりエッジで発生します。$\overline{\text{BG}}$信号はGが変化している間にあるステートに達したあとのクロックの立下りエッジで遷移します。バス制御信号(Tで制御される)は、バスの制御権がMC68030に返されたとき、ステートが変化した直後にプロセッサによってドライブされます。

状態図の最上部中央にあるステート0は、プロセッサがバス・マスタになっている間のバス・アービタのステートで、ここではGとTの両方ともネゲートされています。要求RとアクノリッジAは、それらが両方ともネゲートされている間アービタをステート0に保持します。要求Rを受信すると、許可Gと信号Tの両方ともアサートされます(右上のステート1)。次のクロックにより、左下のステート2に変化し、ここでGとTが保持されます。バス・アービタはアクノリッジAがアサートされるか、要求Rがネゲートされるまでそのステートに留まります。一度、いずれかが発生すると、アービタは中央のステートのステート3に変化し、許可Gをネゲートします。次のクロックはアービタを右上のステート4にしますが、そこで許可Gはネゲートされ、信号Tはアサートされたままになっています。アクノリッジAがアサートされているときは、アービタは、Aがネゲートされるか、要求Rが再びアサートされるまで、ステート4に留まります。Aがネゲートされると、アービタはもとのステートのステート0に戻り、信号Tをネゲートします。このステートのシーケンスは、バスを外部バス・マスタに明け渡すための通常の信号シーケンスに従います。他のステートは、RとAの他の組合せに適用されます。ステート0からステート4の経路で示すとおり、$\overline{\text{BGACK}}$だけを使用して、プロセッサの外部バス・バッファをハイ・インピーダンス状態に置くことができ、単線による調停機能を実現します。

RMWシーケンスは、セマフォ操作と複数プロセッサの同期化をサポートするため、通常は不可分となっています。この不可分シーケンス中、MC68030は$\overline{\text{RMC}}$信号をアサートします。それにより、バス調停ステート・マシンは、RMWシーケンスの最初のリード・サイクル後に発生するバス要求($\overline{\text{BR}}$のアサート)に対して、バス許可を発行($\overline{\text{BG}}$をアサート)しないでそれを無視します。

しかし、場合によっては、RMWシーケンス中にMC68030にバスを放棄させなければならないこ

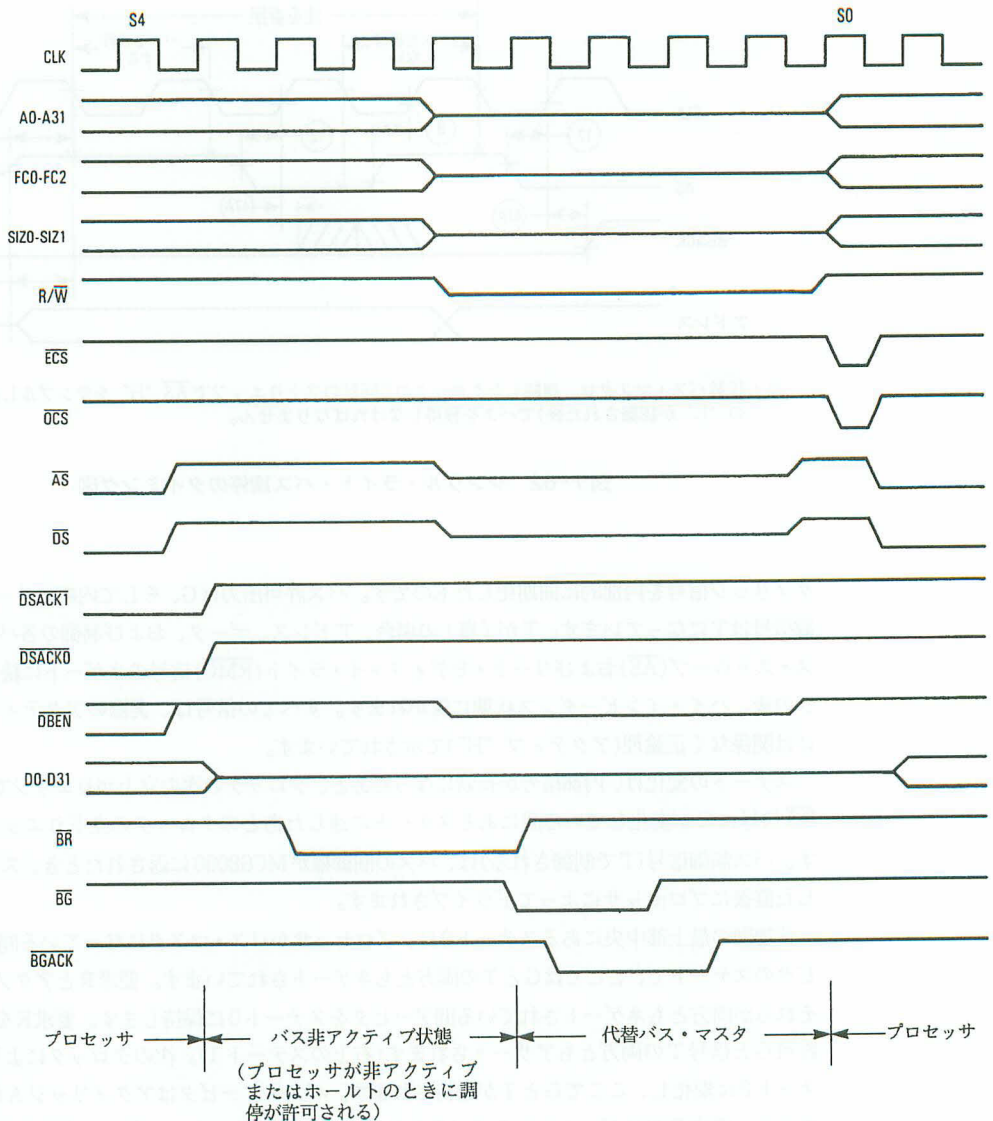



図7-63 バス調停操作(バス非アクティブ)

ともあります。代替バス・マスタがMC68030にバスを放棄させるための方法の1つは、RMWシーケンスの最初のリード・サイクルにのみ適用されます。MC68030では、このリード・サイクル中に通常のバス調停を行なうことができます。これには、通常の放棄と再試行操作(BERR、HALT、およびBRを同時にアサートする)を使用します。なお、この方法はRMWシーケンスの最初のリード・サイクルだけに適用されますが、代替バス・マスタに対しては何の制約も課さないため、RMWの保全性は維持されています。

2番目の方法は単線による調停で、そのタイミングを図7-62に示します。代替マスタは、BGACKをアサートしてMC68030にバスを放棄させ、ASがネゲートされるのを待ってからバスを獲得します。これは、RMWシーケンスのすべてのバス・サイクルに適用されますが、使い方を誤るとシステムの保全性に問題が生じるおそれがあります。代替バス・マスタは、RMWシーケンスでアクセスされるメモリ・ロケーションの内容を変更しないようにして、RMWシーケンスの保全性を保証しなけ

ればなりません。なお、正しく動作させるためには、$\overline{AS}$が連続した2つのクロック・エッジでネゲート("H" になる)されるのを確認してから、代替バス・マスタがバスを獲得するようにする必要があります。この状態を待つことによって、現在のバス動作または保留中のバス動作がすべて完了するか先取りされたことを確認できます。

プロセッサ・バス・サイクルでのバス調停シーケンスのタイミングを図7-60に示します。バスが非アクティブ状態とき(つまり、乗算命令などの内部操作を行なっているとき)のバス調停シーケンスを図7-63に示します。

7. 8　リセット動作

リセット信号($\overline{RESET}$)は、プロセッサまたは外部デバイスがシステムをリセットすることができる双方向性の信号です。システムに電源が投入されると、外部回路はV_{CC}が規定される許容範囲内に入った後、最小100msの間$\overline{RESET}$をアサートしなければなりません。

図7-64は電源投入時におけるリセット動作のタイミング図で、$\overline{RESET}$、V_{CC}、およびバス信号を示します。クロック信号は、V_{CC}が最小動作規定値に達する時点までに安定している必要があります。リセット中は、バス全体(非アクティブ・ステートにドライブできない信号、つまりスリー・ステート不能信号を除く)がスリー・ステート状態になっています。一度、$\overline{RESET}$がネゲートされると、すべての制御信号は非アクティブ状態にドライブされ、データ・バスはリード・モードになり、アドレス・バスがドライブされます。この後、RESET例外処理のための最初のバス・サイクルが始まります。

外部$\overline{RESET}$信号は、プロセッサを含めたシステム全体をリセットします。初期リセットを除いて、$\overline{RESET}$はプロセッサが確実にリセットされるよう、最小520クロック期間はアサートしなければなりません。プロセッサ・ロジック回路をリセットするには、10クロック期間だけ$\overline{RESET}$をアサートすれば十分です。それ以上のクロック期間は、$\overline{RESET}$命令が外部$\overline{RESET}$信号にオーバラップしないようにするためのものです。

プロセッサをリセットすると、進行中のバス・サイクルは、$\overline{DSACKx}$、$\overline{BERR}$、または$\overline{STERM}$がアサートされたときと同様に終了します。さらに、プロセッサはレジスタをリセット例外に合わせて初期化します。リセット操作の例外処理については、「8. 1. 1　リセット例外」で説明しています。

リセット命令が実行されると、プロセッサは512クロック・サイクルの間$\overline{RESET}$信号をドライブします。このとき、プロセッサはシステムの外部デバイスをリセットしますが、プロセッサの内部

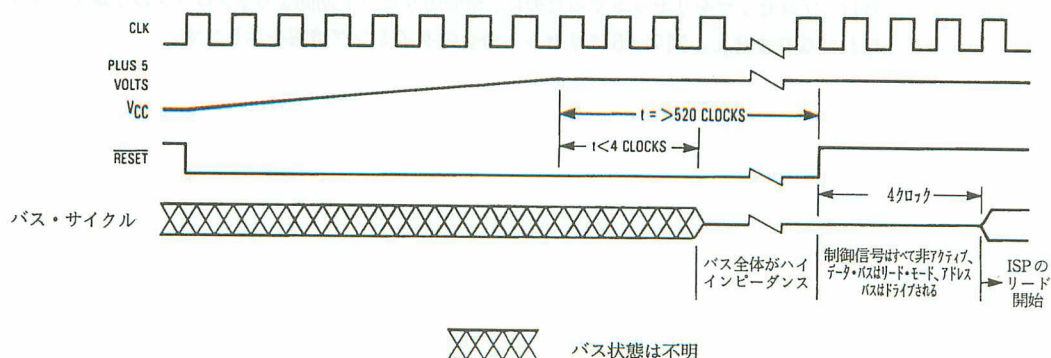



図7-64　初期リセット動作のタイミング

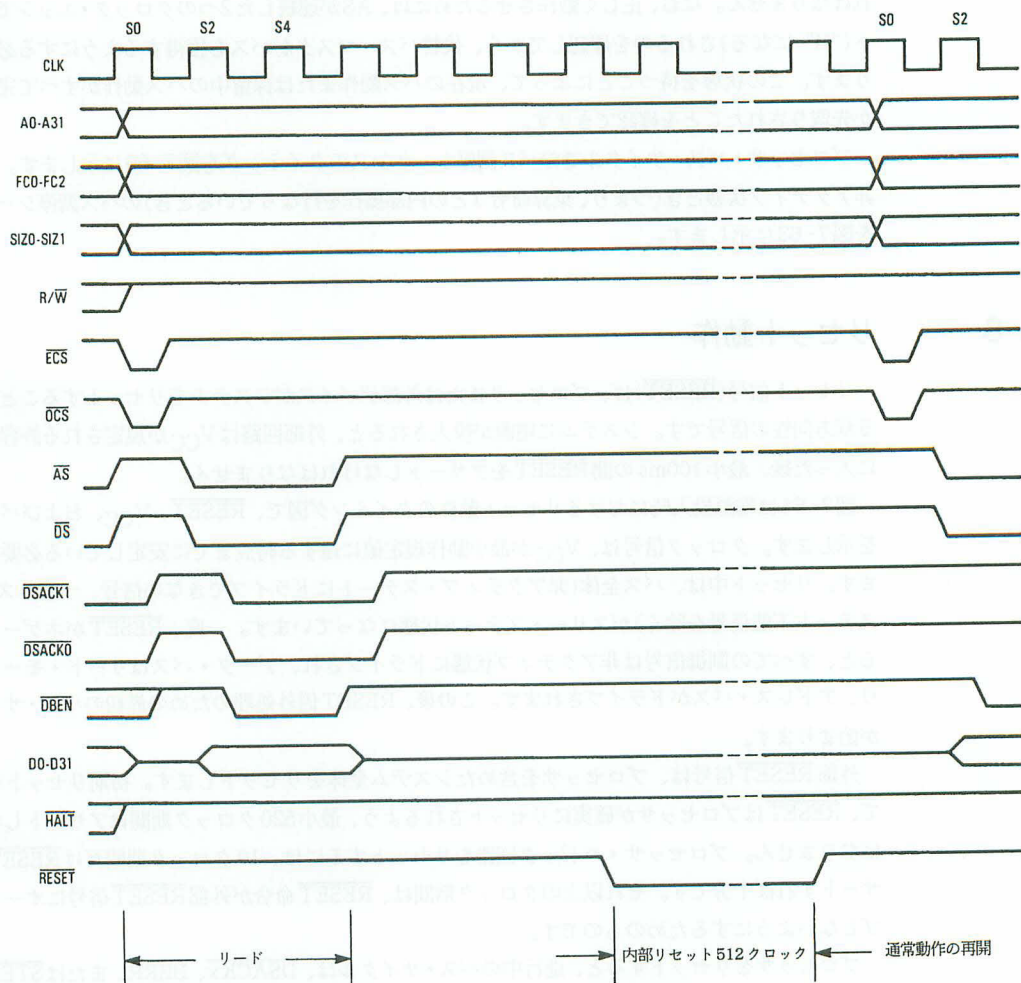



図7-65　リセット命令のタイミング

レジスタは影響を受けません。RESET信号に接続されている外部デバイスは、リセット命令の終了時にリセットされます。リセット命令の実行中にプロセッサに対してアサートされる外部RESET信号は、プロセッサをリセットするために、命令のリセット期間より8クロック以上長くドライブしなければなりません。図7-65にリセット命令のタイミング情報を示します。

第8章

例外処理

例外処理は、例外を引き起こした条件に対するハンドラ・ルーチンの実行に備えて、プロセッサが行なう動作と定義されます。ここで注意したいのは、例外処理には、ハンドラ・ルーチンそのものの実行は含まれていないことです。MC68030プロセッサの処理状態の1つとして例外処理をとらえた説明を、「第4章　処理状態」に記述しています。この章では、各タイプの例外の処理をとりあげながら、例外処理について詳しく説明していきます。また、例外からの復帰およびバス・フォールトの回復についてもふれています。MMUに関連する例外の詳細については、「第9章　メモリ管理ユニット」を参照してください。プロトコルの違反およびコプロセッサ関連の例外の詳細については、「第10章　コプロセッサ・インタフェースの説明」を参照してください。また、浮動小数点コプロセッサに対して定義されている例外の詳細については、MC68881/MC68882のユーザーズ・マニュアルを参照してください。

8.1　例外処理シーケンス

例外処理は、4つの機能ステップで実行されます。しかし、例外処理に関するすべての個別バス・サイクル(ベクタの取得、スタック操作など)は、この章で説明する順序で実行されるとはかぎりません。ただし、スタック・ポインタからのすべてのアドレスおよびオフセットは、ここで説明するとおり扱われることが保証されています。

例外処理の最初のステップでは、ステータス・レジスタが関係します。プロセッサはまずステータス・レジスタの内部コピーを作ります。コピーを作ったあと、Sビットをセットして、スーパバイザ特権状態に切り換えます。次にプロセッサは、T1およびT0ビットをクリアして、例外ハンドラのトレーシングを禁止します。リセットおよび割込み例外の場合は、割込みマスクも更新します。

第2ステップでは、プロセッサは例外のベクタ番号を決定します。割込みの場合、割込みアクノリッジ・サイクルを実行して(CPUのアドレス空間＄Fからの読出し、図7-45および7-46参照)ベクタ番号を取得します。コプロセッサが検出する例外の場合、ベクタ番号はコプロセッサ例外プリミティブ応答の中に含まれています(コプロセッサ例外の詳細については、「第10章　コプロセッサ・インタフェースの説明」を参照してください)。他の例外の場合はすべて、内部回路がベクタ番号を供給します。このベクタ番号を最後のステップで使用して、例外ベクタのアドレスを計算します。なお、この章ではベクタ番号は10進数で表わしています。

リセット以外の例外はすべて、第3のステップでプロセッサの現在のコンテキストがセーブされます。プロセッサは、アクティブ・スーパバイザ・スタックに例外スタック・フレームを1つ生成し、それに例外のタイプに応じたコンテキスト情報を入れます。どの例外が処理されているか、および例外処理実行前のプロセッサの状態によっては、他の情報もスタックされることがあります。例外

が割込みであってステータス・レジスタのMビットがセットされていた場合、プロセッサはMビットをクリアし、割込みスタック内に第2のスタック・フレームを生成します。

最後のステップは、例外ハンドラを始動します。プロセッサはベクタ番号を4倍して、例外ベクタのオフセットを決定します。そして、このオフセットをベクタ・ベース・レジスタに格納されている値に加算して、例外ベクタのメモリ・アドレスを求めます。次にプロセッサは、メモリの例外ベクタ・テーブルからプログラム・カウンタ(およびリセット例外のための割込みスタック・ポインタ(ISP))をロードします。最初の3ワードをプリフェッチして、命令パイプを充てんした後、プロセッサはプログラム・カウンタのアドレスから通常の動作を再開します。表8-1にMC68030で定義されるすべての例外ベクタ・オフセットの説明を示します。

表8-1　例外ベクタの割当て

ベクタ番号	ベクタ・オフセット		割当て	アサートされる STATUS
	Hex	Space		
0	000	SP	リセット初期割込みスタック・ポインタ	–
1	004	SP	リセット初期プログラム・カウンタ	–
2	008	SD	バス・エラー	Yes
3	00C	SD	アドレス・エラー	Yes
4	010	SD	不当命令	No
5	014	SD	ゼロ除算	No
6	018	SD	CHK,CHK2 命令	No
7	01C	SD	cpTRAPcc, TRAPcc, TRAPV 命令	No
8	020	SD	特権違反	No
9	024	SD	トレース	Yes
10	028	SD	ライン1010エミュレータ	No
11	02C	SD	ライン1111エミュレータ	No
12	030	SD	(未定義、予約)	–
13	034	SD	コプロセッサ・プロトコル違反	No
14	038	SD	フォーマット・エラー	No
15	03C	SD	未初期化割込み	Yes
16 〜 23	040 05C	SD SD	未定義、予約	–
24	060	SD	スプリアス割込み	Yes
25	064	SD	レベル1割込みオートベクタ	Yes
26	068	SD	レベル2割込みオートベクタ	Yes
27	06C	SD	レベル3割込みオートベクタ	Yes
28	070	SD	レベル4割込みオートベクタ	Yes
29	074	SD	レベル5割込みオートベクタ	Yes
30	078	SD	レベル6割込みオートベクタ	Yes
31	07C	SD	レベル7割込みオートベクタ	Yes
32 〜 47	080 0BC	SD SD	TRAP ＃0-15 命令ベクタ	No
48	0C0	SD	FPCP 分岐または無秩序状態でのセット	No
49	0C4	SD	FPCP 不正確な結果	No
50	0C8	SD	FPCP ゼロによる除算	No
51	0CC	SD	FPCP アンダフロー	No
52	0D0	SD	FPCP オペランド・エラー	No
53	0D4	SD	FPCP オーバフロー	No
54	0D8	SD	FPCP シグナル NAN	No
55	0DC	SD	未定義、予約	No
56	0E0	SD	MMUコンフィギュレーション・エラー	No
57	0E4	SD	MC68851を定義、MC68030では使用しない	–
58	0E8	SD	MC68551を定義、MC68030では使用しない	–
59 〜 63	0EC 0FC	SD SD	未定義、予約	–
64 〜 255	100 255	SD SD	ユーザ定義ベクタ(192)	Yes

SP＝スーパバイザ・プログラム空間
SD＝スーパバイザ・データ空間

表8-2　マイクロシーケンサの$\overline{\text{STATUS}}$表示

アサートされるタイミング	表　　示
1 クロック	命令境界でのシーケンサ――次の命令で実行を開始
2 クロック	命令境界でのシーケンサ、ただし次の理由により次の命令で実行を開始しない。 ●トレース例外の保留 　または ●割込み例外の保留
3 クロック	MMUアドレス変換キャッシュ・ミス――プロセッサがテーブル・サーチを開始 　または 次のいずれかの条件により例外処理を開始 ●リセット ●バス・エラー ●アドレス・エラー ●スプリアス割込み ●オートベクタ割込み ●F系列命令(コプロセッサの応答なし)
連続	ダブル・バス・フォールトのためにプロセッサが停止

表8-1に示すように、最初の64個のベクタはモトローラによって定義されており、192のベクタがユーザ定義の割込みベクタとして確保されています。ただし、システム設計者の責任において、内部目的のために確保されているベクタを、外部デバイスが使用することもできます。

MC68030は$\overline{\text{STATUS}}$信号によって、命令の境界といくつかの例外を識別します。表8-2に示すように、$\overline{\text{STATUS}}$は内部マイクロシーケンサの状態に応じて、命令の境界と例外を示します。さらに、$\overline{\text{STATUS}}$はMMUがアドレス変換キャッシュ・ミスが発生し、プロセッサがミスの原因となった論理アドレスへのテーブル・サーチ・アクセスを開始しようとしていることを知らせます。表8-1に示すように、命令関連例外は$\overline{\text{STATUS}}$をアサートしません。$\overline{\text{STATUS}}$信号のタイミング情報については、「第12章　アプリケーション情報」を参照してください。

8. 1. 1　リセット例外

$\overline{\text{RESET}}$信号の外部ハードウェアをアサートすると、リセット例外が発生します。$\overline{\text{RESET}}$をアサートするのに必要な条件については、「7. 8　リセット動作」を参照してください。

リセット例外は例外の中で最も優先順位が高くなっています。リセット信号はシステムの初期化および致命的な故障から復帰するために使用します。リセットを認識したときに実行されていた処理はすべてアボートされ、回復することはできません。図8-1はリセット例外のフローチャートで、以下の操作を実行します。

1. ステータス・レジスタの両方のトレース・ビットをクリアして、トレースをディセーブルします。
2. ステータス・レジスタのスーパバイザ・ビットをセットし、マスタ・ビットをクリアして、プロセッサをスーパバイザ特権レベルの割込みモードにします。
3. プロセッサの割込み優先マスクを最高優先レベル(レベル7)に設定します。
4. ベクタ・ベース・レジスタをゼロ($ 00000000)に初期化します。
5. 両方のオンチップ・キャッシュのイネーブル、凍結、およびバースト・イネーブル・ビット、そしてキャッシュ制御レジスタのデータ・キャッシュに対応するライト・アロケート・ビットをクリアします。
6. 命令およびデータ・キャッシュ内のすべてのエントリを無効にします。
7. 変換制御レジスタのイネーブル・ビットとMMUの両方のトランスペアレント変換レジスタのイネーブル・ビットをクリアします。

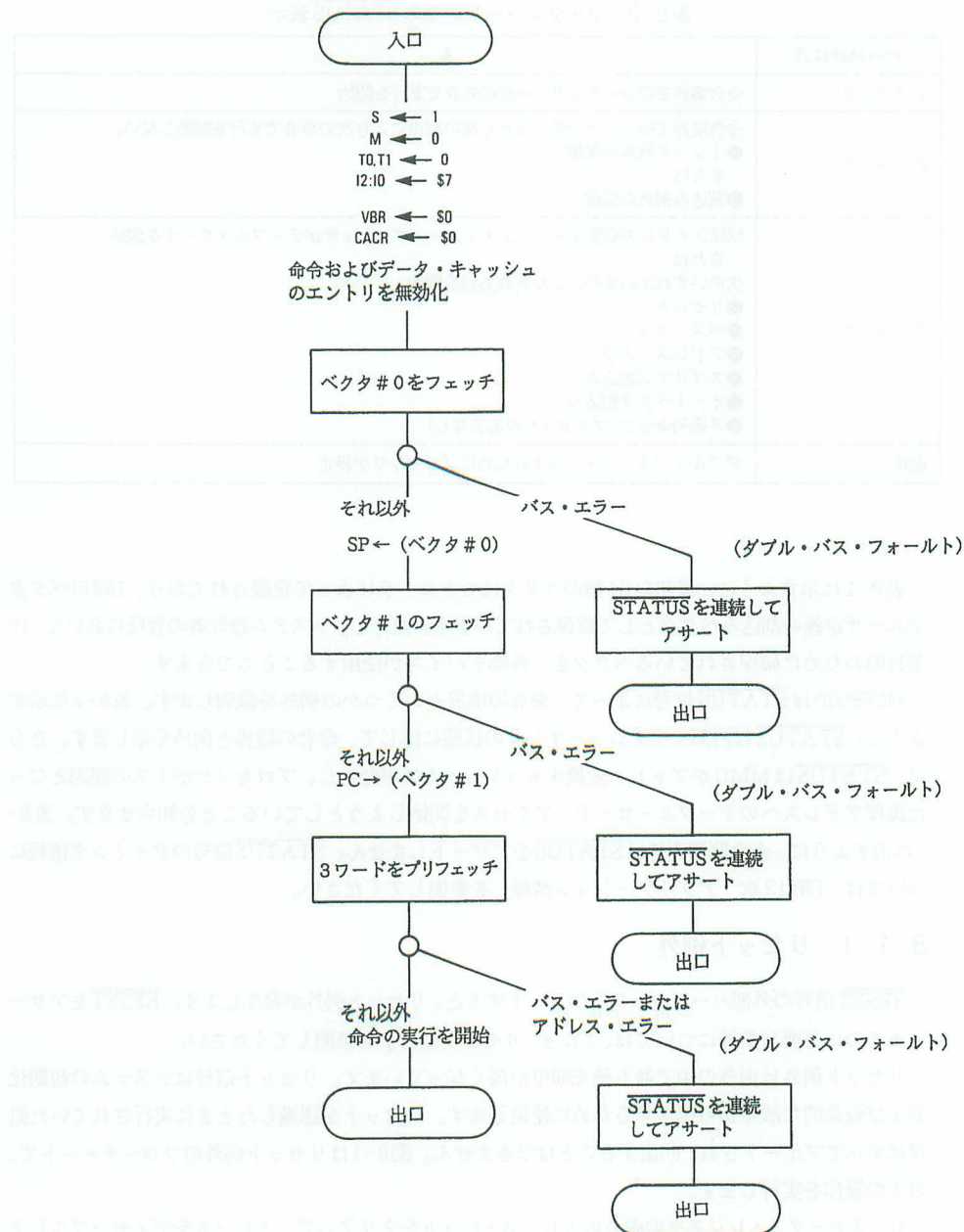



図8-1　リセット操作のフローチャート

8. スーパバイザ・プログラム・アドレス空間のオフセット・ゼロにおいて、リセット例外ベクタ(2ロング・ワード)を参照するベクタ番号を生成します。
9. リセット例外ベクタの第1ロング・ワードを割込みスタック・ポインタにロードします。
10. リセット例外ベクタの第2ロング・ワードをプログラム・カウンタにロードします。

初期命令プリフェッチの後、プログラムの実行はプログラム・カウンタのアドレスから開始されます。リセット例外はアドレス変換キャッシュ(ATC)をフラッシュせず、またプログラム・カウンタやステータス・レジスタの値をセーブすることもありません。

「7.5.4　ダブル・バス・フォールト」で説明したように、リセットに対する例外処理シーケンス

の実行中にバス・エラーまたはアドレス・エラーが発生した場合は、ダブル・バス障害が発生します。すると、プロセッサは停止し、$\overline{STATUS}$信号が継続的にアサートされ、ホルト状態を示します。

リセット命令の実行によってリセット例外が発生することはなく、また内部レジスタに影響を与えることもありませんが、MC68030は$\overline{RESET}$信号をアサートして、すべての外部デバイスをリセットします。

8. 1. 2 バス・エラー例外

バス・エラー例外は、外部回路が$\overline{BERR}$入力信号をアサートして、バス・サイクルをアボートしたときに発生します。アボートされたバス・サイクルがデータ空間をアクセスしていた場合、プロセッサは直ちに例外処理を開始します。命令プリフェッチのバス・サイクルがアボートされた場合、プロセッサはプリフェッチした情報が必要になるときまで、例外の処理を延期することができます。バースト操作の第2、第3、または第4アクセス中に$\overline{BERR}$信号をアサートしてもバス・エラー例外とはなりませんが、バーストはアボートされます。バースト操作中のバス・エラーの影響については、「6. 1. 3. 2 バースト・モードの充てん」および「7. 5. 1 バス・エラー」を参照してください。

バス・エラー例外は、MMUがアドレス変換を正常に実行できないことを検出したときにも発生します。さらに、ATCミスが発生し外部バス・サイクルが要求されると、MMUはそのバス・サイクルをアボートし、メモリの変換テーブルでマッピングをサーチし、バス・サイクルを再試行します。テーブル・サーチ中に発生した問題のためにその論理アドレスに対する変換が有効でない場合(そのページの変換テーブルにある適当なページ・ディスクリプタへのアクセスの試行)、アボートされたバス・サイクルを再試行したときにバス・エラー例外が発生します。

発生する問題には、リミット違反、無効ディスクリプタ、または変換テーブルをアクセスするのに使用するバス・サイクル中での$\overline{BERR}$信号のアサートがあります。ATCでミスが起こると、プロセッサは自動的にテーブル・サーチを開始しますが、前述した特定の条件の1つが発生しないかぎり、バス・エラー例外とはなりません。

プロセッサは現在のステータス・レジスタのコピーを作成して、バス・エラーの例外処理を開始します。続いてプロセッサは、スーパバイザ特権レベルに入り(ステータス・レジスタのSビットをセットして)、トレース・ビットをクリアします。プロセッサはバス・エラー・ベクタに対し、例外ベクタ番号2を生成します。そして、ベクタ・オフセット、プログラム・カウンタ、およびステータス・レジスタの内部コピーをスタックにセーブします。セーブされるプログラム・カウンタ値は、そのフォールトが検出されたときに実行中であった命令の論理アドレスです。これは必ずしもそのバス・サイクルを開始した命令ではありません。というのは、プロセッサは命令をオーバラップして実行するからです。プロセッサは一部の内部レジスタの内容もセーブします。スタックにセーブされる情報は、バス・フォールトの原因を知ってそのエラーから回復するのに十分なものです。

効率を向上させるために、MC68030は2種類のバス・エラー・スタック・フレーム・フォーマットを使用しています。1つの命令の境界でバス・エラーが発生した場合は、そのエラーから回復するにはわずかな情報があればよく、プロセッサは表8-7に示すような、短いバス・フォールト・スタック・フレームを構築します。命令の実行途中でバス・エラーが発生した場合、プロセッサはエラーからの回復に備えて、全体の状態をセーブし、表8-7に示すロング・バス・フォールト・スタック・フレームを使用します。

スタック・フレームのフォーマット・コードでこの2つのスタック・フレーム・フォーマットを区別しています。スタック・フレーム・フォーマットの詳細については、「8. 4 例外スタック・フレーム・フォーマット」を参照してください。

バス・エラー、アドレス・エラー、またはリセットの例外処理中、あるいはプロセッサがRTE 命

令の実行中にスタックから内部状態情報をロードしている間に、バス・エラーが発生した場合はダブル・バス・フォールトとなり、プロセッサはホルト状態に入ります。この状態は$\overline{\text{STATUS}}$信号を継続してアサートすることによって表示されます。この場合、プロセッサは現在のメモリの状態を変更することはありません。ダブル・バス障害でホルト状態になったプロセッサを再スタートさせるには、外部$\overline{\text{RESET}}$を使用するしかありません。

8.1.3　アドレス・エラー例外

アドレス・エラー例外は、プロセッサが奇数アドレスから命令をプリフェッチしようとしたときに発生します。バス・エラー例外とよく似ていますが、内部で発生するものです。バス・サイクルは実行されず、プロセッサは直ちに例外処理を開始します。例外処理が開始されたあとのシーケンスは、ベクタ番号が3でスタック・フレームのベクタ・オフセットがアドレス・エラー・ベクタを参照する以外は、「8.1.2　バス・エラー例外」で説明したバス・エラー例外の場合と同じです。ショートまたはロング・バス・フォールト・スタック・フレームのいずれかが生成されます。また、バス・エラー、アドレス・エラー、またはリセットに対する例外処理の実行中にアドレス・エラーが発生した場合、ダブル・バス・フォールトになります。

8.1.4　命令トラップ例外

特定の命令を使用すれば、意図的にトラップ例外を発生させることができます。TRAP＃n命令は常に例外を発生しますので、ユーザ・プログラムでシステム・コールを行なうのに使用すると便利です。TRAPcc、TRAPV、cpTRAPcc、CHKおよびCHK2命令は、ユーザ・プログラムが、演算オーバフローや境界値の限界超過などのエラーを検出すると例外を発生します。

DIVS命令およびDIVU命令では、0による除算を実行しようとすると例外が発生します。

トラップ例外が発生すると、プロセッサは内部でステータス・レジスタをコピーし、スーパバイザ特権レベルに入り、トレース・ビットをクリアします。トラップの原因となった命令に対してトレースがイネーブルされていた場合、トラップ・ハンドラからのRTE命令の後、トレース例外が発生し、トレースはトラップ命令に一致します。トラップ・ハンドラ・ルーチンはトレースされません。プロセッサは実行中の命令に応じてベクタ番号を生成します。TRAP＃n命令の場合、ベクタ番号は32＋nになります。スタック・フレームは、トラップ・ベクタ・オフセット、プログラム・カウンタ、およびステータス・レジスタの内部コピーをスーパバイザ・スタックにセーブします。セーブされるプログラム・カウンタの値は、トラップを発生した命令の次に実行される命令の論理アドレスです。TRAP＃nを除くすべての命令トラップの場合、トラップを発生した命令に対するポインタもセーブされます。最後に、必要な命令プリフェッチのあと、例外ベクタのアドレスから命令の実行を再開します。

8.1.5　不当命令または未実装命令例外

MC68030 の有効な命令の第1ワードのビット・パターンのどれにも該当しないビット・パターンをもつ命令、または最初の拡張ワードのレジスタ指定フィールドが定義されていないMOVEC命令を不当命令(illegal　instruction)とよびます。

不当命令例外は、ブレークポイント・アクノリッジ・サイクル(「7.4.2　ブレークポイント・アクノリッジ・サイクル」で説明する)が、バス・エラー信号のアサートによって終了した場合に発生します。これは、外部回路が命令パイプの中で、BKPT命令ワードを置き換える命令を供給しなかったことを意味します。

ビット［15：12］が＄Aに等しいワード・パターンをもつ命令ワードは、A系列のオペコードをもつ未実装命令(unimplemented　instruction)とよびます。プロセッサがA系列のオペコードをも

つ未実装命令を実行しようとすると、ベクタ番号10の例外が生成されるため、未実装命令を効率よくエミュレートすることができます。

ビット［15：12］が＄F、ビット［11：9］が＄0に等しいワード・パターンをもち、後続ワードのワード・パターンが定義されている命令は、正当なMMU命令です。ビット［15：12］が＄Fで、ビット［11：9］が＄0に等しいけれども、後続ワードのワード・パターンが未定義の命令は、スーパバイザ・モードで実行しようとした場合は、F系列のオペコードをもつ未実装命令として扱われます。このような命令をユーザ・モードで実行しようとした場合は、特権違反例外が発生します。F系列オペコードをもつ未実装命令の例外ベクタ番号は11です。

ビット［15：12］が＄Fで、ビット［11：9］が＄0以外のワード・パターンは、コプロセッサ命令として使用されています。プロセッサがコプロセッサ命令を識別すると、CPU 空間のタイプ＄2（「4. 2　アドレス空間の種類」）を参照するバス・サイクルを実行し、7つのコプロセッサの1つをアドレス指定します（ビット［11：9］に応じて1-7）。アドレス指定されたコプロセッサがシステムに存在せず、そのアクセスがバス・エラーで終了した場合、その命令は未実装命令（F系列のオペコード）となります。システムはF系列の例外ハンドラを使用してコプロセッサの機能をエミュレートすることができます。詳細については、「第10章　コプロセッサ・インタフェースの説明」を参照してください。

不当命令および未実装命令に対する例外処理は、命令トラップの場合とよく似ています。プロセッサが不当命令または未実装命令を識別すると、その命令を実行しようとしないで例外処理を開始します。プロセッサはステータス・レジスタをコピーして、スーパバイザ特権レベルになり、トレース・ビットをクリアしてトレースをディセーブルします。プロセッサは例外のタイプに応じて、4、10、または11のベクタ番号を生成します。不当命令または未実装命令のベクタ・オフセット、プログラム・カウンタの現在値、およびステータス・レジスタのコピーがスーパバイザ・スタックにセーブされます。セーブされたプログラム・カウンタの値は、その不当命令または未実装命令のアドレスです。最後に、例外ベクタにあるアドレスから命令の実行を再開します。命令をソフトウェアでエミュレートしたり、ハンドラからの戻りでスキップする場合に、スタックされたプログラム・カウンタを調整するのは、処理ルーチンの責任です。

8. 1. 6 特権違反例外

システムの安全性を確保するために、表8-3に掲載する命令が特権化されています。ユーザ特権レベルで、これらの特権命令の1つを実行しようとすると例外が発生します。また、コプロセッサが特権状態のチェックを要求したときに、プロセッサがユーザ・レベルにあるときにも特権違反が発生します。

表8-3　特権命令

ANDI TO SR	ORI to SR
EOR to SR	PFLUSH
cpRESTORE	PLOAD
cpSAVE	PMOVE
MOVE from SR	PTEST
MOVE to SR	RESET
MOVE USP	RTE
MOVEC	STOP
MOVES	

特権違反に対する例外処理は、不当命令の場合とよく似ています。プロセッサが特権違反を識別すると、その命令を実行する前に例外処理を開始します。プロセッサはステータス・レジスタをコピーし、スーパバイザ特権レベルに入り、トレース・ビットをクリアします。プロセッサは特権違反例外ベクタであるベクタ番号8を生成し、特権違反ベクタ・オフセット、プログラム・カウンタの現在値、およびステータス・レジスタの内部コピーをスーパバイザ・スタックにセーブします。セーブされるプログラム・カウンタの値は、その特権違反を起こした命令の第1ワードの論理アドレスです。最後に、特権違反の例外ベクタにあるアドレスから必要なプリフェッチを行なった後命令の実行を再開します。

8. 1. 7　トレース例外

プログラム開発を支援するために、M68000プロセッサは命令ごとにトレースを行なう機能を備えています。MC68030 ではすべての命令をトレースしたり、プログラムの流れを変える命令だけをトレースすることが可能です。トレース・モードでは命令が実行された後、トレース例外が発生するため、デバッグ・プログラムでテスト中のプログラムの実行をモニタできます。

ステータス・レジスタのスーパバイザ部分にあるT1およびT0のビットでトレースを制御します。命令が実行を開始したときのこれらのビットの状態で、命令が完了した後トレース例外を発生するか否かが決まります。T1およびT0を両方ともクリアした場合、トレースはディセーブルされ、命令は通常どおり実行されます。T1ビットをクリアし、T0ビットをセットすると、プログラムの流れを変える命令だけがトレース例外を発生します。プログラム・カウンタをインクリメントする命令は、通常トレース例外を発生しません。このモードでトレースされる命令には、すべて分岐、ジャンプ、命令トラップ、およびプログラム・カウンタの流れを変更するコプロセッサ命令があります。このモードにはステータス・レジスタの操作が含まれていますが、これはステータス・レジスタを変更する可能性のある命令を実行したときには、プロセッサは再度パイプを充てんするために、命令ワードを再プリフェッチしなければならないためです。BKPT命令を実行すると、BKPT命令を置き換えるオペコードがプログラムの流れを変更する命令(つまり、ジャンプ、分岐など)の場合は、プログラムの流れが変わります。T1ビットをセットし、T0ビットをクリアした場合、すべての命令の実行によって、トレース例外が発生します。表8-4にビットT1とT0の組合せによって選択されるトレース・モードを示します。

一般に、トレース例外はトレースされる任意の命令の機能拡張といえます。つまり、トレースされた命令の実行は、トレース例外処理が終わるまで終了しないということです。バス・エラーまたはアドレス・エラー例外のために、命令の実行が終了しない場合、中断されていた命令が再開され、通常どおり実行が終了するまで、トレース例外処理は延期されます。命令の終了時に、割込みが保留されていた場合は、トレース例外処理が終了してからその割込み例外が処理されます。命令が通常の処理の中で強制的に例外を発生するようになっている場合は、トレース例外を処理する前に強制的な例外処理が発生します。例外の優先順位に関する詳細については、「8. 1. 12　多重例外」を参照してください。

プロセッサがトレース・モードになっているときに、不当命令または未実装命令を実行しようとした場合、その命令は実行されないためトレース例外は発生しません。これは命令の機能を実行し、スタックされたプログラム・カウンタを変更して未実装命令をスキップしてからリターンする、命令エミュレーション・ルーチンにとっては特に重要です。リターンする前に、スタック内のステータス・レジスタのトレース・ビットをチェックしなければなりません。そしてトレースがイネーブルされていた場合は、トレース例外ハンドラがエミュレートされている命令の結果も表示するように、トレース例外処理もエミュレートしなければなりません。

表8-4　トレースの制御

TI	T0	トレース機能
0	0	トレースなし
0	1	フローの変化をトレース(BRA, JMPなど)
1	0	命令の実行をトレース(すべての命令)
1	1	未定義、予約

トレースの例外処理は、トレースした命令を通常どおり処理し、次の命令を開始する前に開始されます。プロセッサはまずステータス・レジスタの内部コピーを作成し、スーパバイザ特権レベルに入ります。ステータス・レジスタのT1およびT0ビットをクリアし、それ以上トレースを行なわないようにします。プロセッサはトレース例外に対してベクタ番号9を供給し、トレース例外ベクタ・オフセット、プログラム・カウンタの現在値、およびステータス・レジスタのコピーをスーパバイザ・スタックにセーブします。セーブされるプログラム・カウンタの値は、次に実行する命令の論理アドレスです。命令の実行はトレース例外ベクタにあるアドレスから、必要なプリフェッチを行なったあとで再開されます。

STOP命令はトレースされているときには、その機能を実行しません。T1＝1およびT0＝0の状態でSTOP命令の実行を開始した場合は、そのSTOP命令がステータス・レジスタに値をロードした後、トレース例外が発生します。トレース・ハンドラ・ルーチンから戻ると、STOP命令の次の命令から実行を継続するため、プロセッサがストップ状態になることはありません。

8．1．8　フォーマット・エラー例外

プリフェッチされた命令が有効かどうかをチェックするのと同様に、プロセッサは(必要に応じ、コプロセッサの助けを借りて)制御動作のためのデータ値のチェックを行ないます。この例としては、cpRESTORE 命令のためのコプロセッサのステート・フレーム・フォーマット、そしてRTE命令のためのスタック・フレームのフォーマットのチェックがあります。

RTE命令はスタック・フォーマット・コードの妥当性をチェックし、ロング・バス・サイクル・フォールト・フォーマット・フレームの場合は、プロセッサの内部バージョン番号をメモリ・ロケーションSP＋54(SP＋＄36)にあるフレームと比較します。このチェックによって、プロセッサがスタック・フレームの内部状態情報を正しく解釈できることが保証されます。

cpRESTORE 命令は、コプロセッサのステート・フレームのフォーマット・ワードを妥当性チェックのためにコプロセッサに渡します。コプロセッサがそのフォーマット値を正当なものと判断しなかった場合は、MC68030にフォーマット・エラー例外処理を実行するよう通知します。コプロセッサ関連の例外の詳細については、「第10章　コプロセッサ・インタフェースの説明」を参照してください。

上記のチェックのいずれかによって、スタックされたデータのフォーマットが不適切であると分かった場合、命令はフォーマット・エラー例外を発生します。この例外はショート・フォーマットのスタック・フレームをセーブしてから、例外ベクタ番号14を生成し、フォーマット例外ベクタにあるアドレスから実行を継続します。スタックされたプログラム・カウンタ値は、フォーマット・エラーを検出した命令の論理アドレスです。

8．1．9　割込み例外

周辺デバイスがMC68030のサービスを要求するか、プロセッサが必要とする情報を送信できる状態になると、プロセッサに割込み例外処理を実行するよう通知することができます。割込み例外は、適切に応答するルーチンにも制御を渡します。

周辺デバイスは“L”アクティブの割込み優先レベル信号($\overline{IPL0}$-$\overline{IPL2}$)を使用して、プロセッサに割込み条件を通知し、その条件の優先レベルを指定します。3つの信号は0～7の値をエンコードします($\overline{IPL0}$が最下位ビット)。3つの信号がすべて“H”レベルの場合は、割込み要求なし(レベル0)に相当し、$\overline{IPL0}$-$\overline{IPL2}$に“L”レベルがあると割込み要求のレベル7に相当します。1～7の値は優先割込みの1～7を指定します。レベル7が最も高い優先順位です。外部回路は各レベルのデバイスからの信号をチェインまたはマージし、プロセッサに割込みを要求するデバイス数に制限はありません。

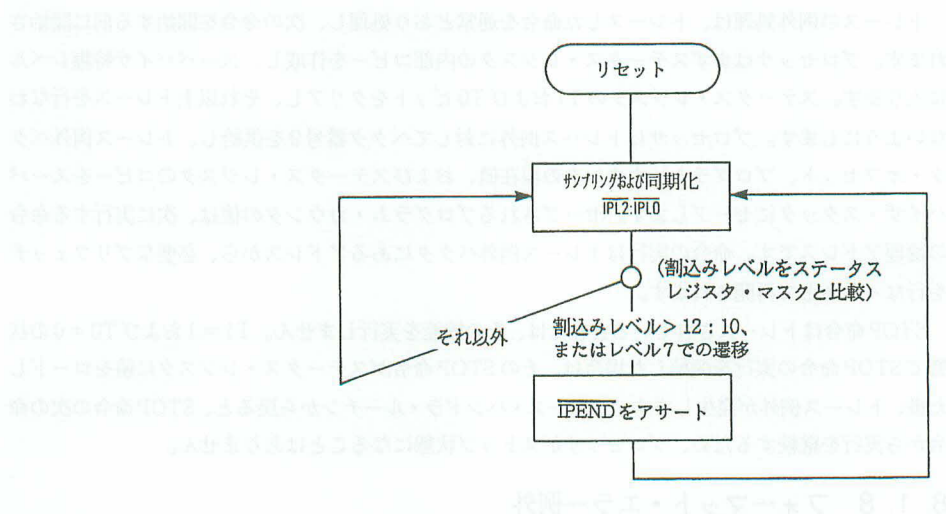



図8-2　割込み保留手順

$\overline{IPL0}$-$\overline{IPL2}$の割込み信号はMC68030が、割込みが認識されるのを保証するために、割込みを認識応答するまで、割込み要求レベルを維持しなければなりません。MC68030はプロセッサ・クロックの立下りエッジで$\overline{IPL0}$-$\overline{IPL2}$信号を継続的にサンプルして、これらの信号の同期化とデバウンスを行ないます。2連続クロック期間にわたって安定している割込み要求は有効な入力とみなされます。プロトコルでは、プロセッサがその割込み値に対応した割込みアクノリッジ・サイクルを実行するまで、要求が保持されていることを求めていますが、最小2クロック・サイクルの長さの割込み要求であれば認識されるはずです。

MC68030のステータス・レジスタには、割込み優先マスク(I2、I1、I0、ビット10-8)があります。割込みマスクの値はプロセッサが無視する最も高い優先レベルです。割込み要求の優先順位が、このマスクの値より高い場合、プロセッサはその要求を保留割込みにします。図8-2に割込みを保留にするための手順のフローチャートを示します。

複数のデバイスが同じ割込みレベルに接続されていた場合、各デバイスは、すべての要求が確実に処理されるために、対応する割込みアクノリッジ・サイクルの間、割込み優先レベルを一定に保持しなければなりません。

表8-5に割込みレベル、各レベルを定義するIPL2-IPL0、および各レベルにおける割込みを許可するマスク値を一覧にして示します。

表8-5　割込みレベルとマスク値

要求割込みレベル	制御ライン・ステータス			認識に必要な割込みマスク・レベル
	$\overline{IP2}$	$\overline{IP1}$	$\overline{IP0}$	
0	"H"	"H"	"H"	N/A*
1	"H"	"H"	"L"	0
2	"H"	"L"	"H"	0-1
3	"H"	"L"	"L"	0-2
4	"L"	"H"	"H"	0-3
5	"L"	"H"	"L"	0-4
6	"L"	"L"	"H"	0-5
7	"L"	"L"	"L"	0-7

*割込みが必要ないことを示す。

割込みレベル7のノンマスカブル割込み(NMI)は特殊なケースです。レベル7の割込みは、割込み優先マスクでマスクすることはできず、またレベルの遷移に感応します。プロセッサは、マスクの値に関係なく外部割込み要求レベルが、下位レベルからレベル7へ変化するたびに割込み要求を認識します。図8-3に割込みを認識する例を2とおり示します。1つはレベル6でもう1つはレベル7です。MC68030がレベル6の割込みを処理すると、ハンドラ・ルーチンに入る前にステータス・レジスタのマスクが自動的にレベル6で更新されるため、それ以降のレベル6の割込みはマスクされます。このマスク値を下げるような命令が実行されなければ、外部要求をレベル3に下げてから、レベル6に戻せば、2番目のレベル6の割込みは処理されません。しかし、MC68030がレベル7の割込み(ステータス・レジスタのマスクを7に設定)を処理していて、外部要求をレベル3に下げてからレベル7に戻した場合は、2番目のレベル7の割込みが処理されます。2番目のレベル7の割込みが処理されるのは、レベル7の割込みがレベル・センシティブであるためです。レベル7の割込みは、要求レベルとマスク・レベルが7になっているときに、割込み優先マスクのレベルをそれ以下にセットしたとき(たとえば、MOVE to SR命令またはRTE命令によって)にも発生します。図8-3のレベル6の割込み要求レベルおよびマスク・レベルに示すとおり、これがすべての割込みレベルの場合です。

なお、マスク値6およびマスク値7の両方とも、1～6の要求レベルの認識を禁止します。これらの唯一の違いは割込み要求レベルが7で、マスク値が7のときに発生します。マスク値が6に下げられると、2番目のレベル7の割込みが認識されます。

MC68030は割込み要求を保留にするときには、割込み保留信号($\overline{\text{IPEND}}$)をアサートします。図8-4にIPLラインでの割込みレベルのアサートに対する$\overline{\text{IPEND}}$のアサートを示します。$\overline{\text{IPEND}}$は、次の命令境界で(優先順位の高い例外に続いて)割込み例外が発生することを外部デバイスに知らせます。

$\overline{\text{IPEND}}$信号の状態はバス操作に関係なく、各命令ごとに1回ずつ、内部でプロセッサがチェックします。また、例外処理に関連する2番目の命令のプリフェッチ中にもチェックされます。図8-5に割込みの認識および関連の例外処理シーケンスのフローチャートを示します。

保留割込みを処理するときに、命令の境界を予測するために、その割込みに対する$\overline{\text{IPEND}}$のアサートと$\overline{\text{STATUS}}$のアサート間のタイミング関係を調べなければなりません。図8-6に2つの割込み認識の例を示します。$\overline{\text{IPEND}}$の後の最初の$\overline{\text{STATUS}}$のアサートはSTAT0で表わします。次の$\overline{\text{STATUS}}$のアサートをSTAT1で表わします。

STAT0が$\overline{\text{IPEND}}$をアサートさせたクロック・エッジの直後のクロックの立下りエッジで開始すると(例1に示す)、STAT1は最小2クロック長となり、ほかに保留されている例外がない場合は、STAT1で定義される境界で割込みが認識されます。$\overline{\text{IPEND}}$がSTAT0までアサートされた場合、STAT0で定義された境界で割込みが認識される場合があります(例2に示す)。その場合、STAT0が2クロックの間アサートされてこの状態を知らせます。

同期化された上位の優先割込みがない場合、割込みアクノリッジ・サイクルのステート0(S0)で、$\overline{\text{IPEND}}$信号がネゲートされ「7. 4. 1. 1　割込みアクノリッジ・サイクル――通常終了」参照)、この時点でアクノリッジ中の$\overline{\text{IPLx}}$信号をネゲートすることができます。

割込み例外の処理を実行するとき、プロセッサはまずステータス・レジスタの内部コピーを作成し、特権状態をスーパバイザに設定し、トレースを抑止して、プロセッサの割込みマスク・レベルをサービス中の割込みレベルに設定します。プロセッサは割込みアクノリッジ・バス・サイクルを使用して、アドレス・バスのピンA1～A3に出力されている割込みレベル番号により、割込みを要求しているデバイスからベクタ番号を取得します。ベクタ番号を供給できないデバイスの場合、自動ベクタ信号($\overline{\text{AVEC}}$)をアサートすることができ、MC68030は内部で生成される割込みレベル番号に対応した25-31のベクタ番号のいずれかを使用します。割込みアクノリッジ・サイクル中に外部回路がバス・エラーを表示した場合、その割込みはスプリアス(spurious)とみなされ、プロセッサ

外部$\overline{\mathrm{IPL2}}$：$\overline{\mathrm{IPL0}}$　　SRマスク（I2：I0）　　アクション

レベル6の例：

100 ($3)　　101 ($5)　　初期状態

001（＄6）の場合、110（＄6）およびレベル6の割込み　　（レベルの比較）

100（＄3）で依然として110（＄6）の場合、何もしない

001（＄6）で依然として110（＄6）の場合、何もしない

依然として001（＄6）でRTEの場合、101（＄5）となり、レベル6の割込み　　（レベルの比較）

レベル7の例：

100 ($3)　　101 ($5)　　初期状態

000（＄7）の場合、111（＄7）およびレベル7の割込み　　（遷移）

100（＄3）で依然として111（＄7）の場合、何もしない

000（＄7）で依然として111（＄7）の場合、レベル7の割込み　　（遷移）

依然として000（＄7）でRTEの場合、101（＄5）となり、レベル7の割込み　　（レベルの比較）

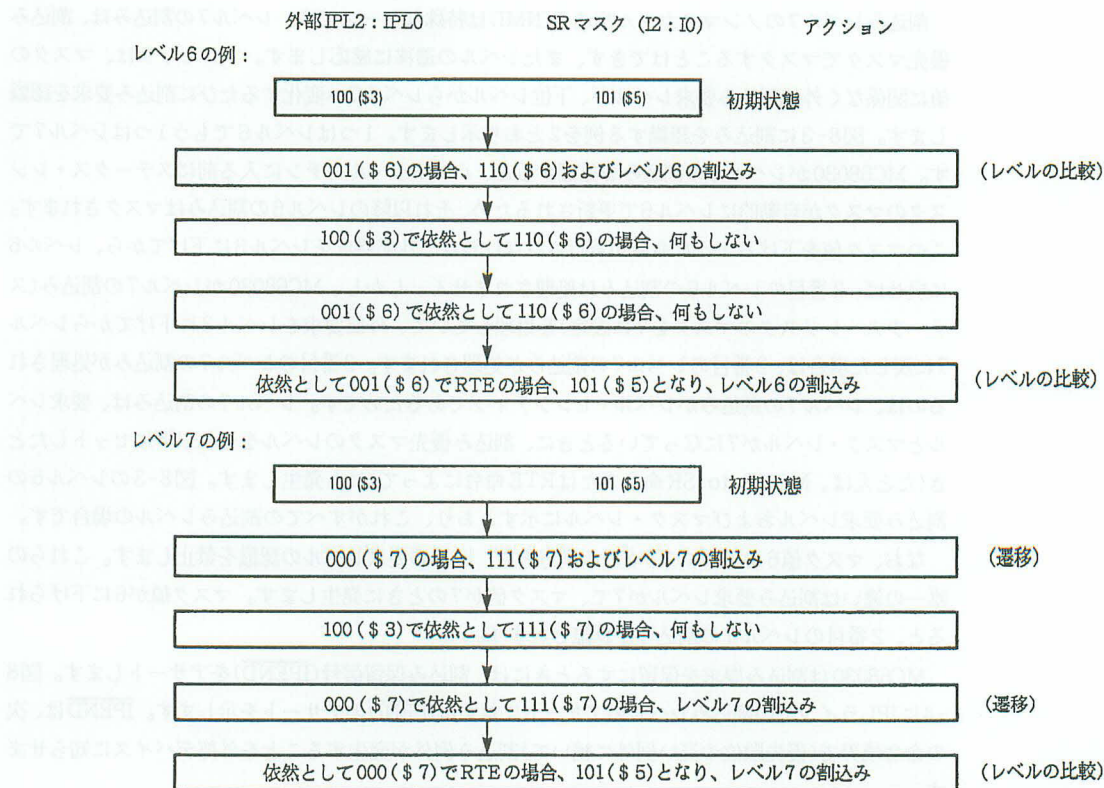



図8-3　割込み認識の例

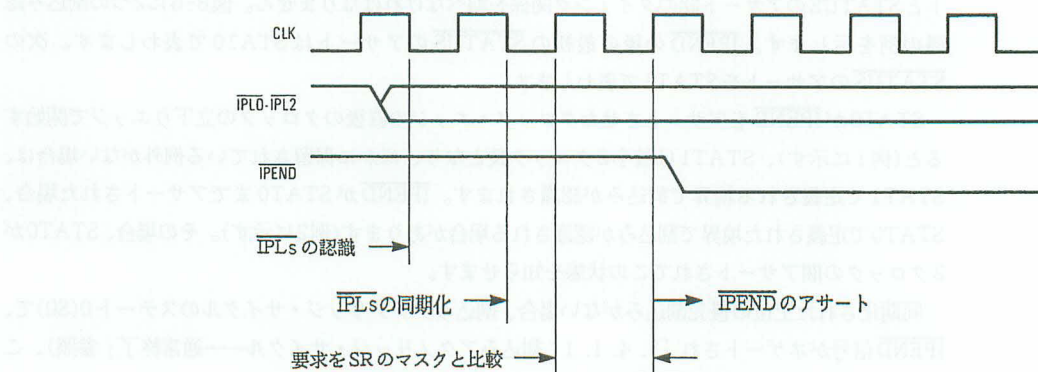



図8-4　$\overline{\mathrm{IPEND}}$のアサート

はスプリアス割込みベクタ番号24を生成します。割込みバス・サイクルの詳細については、「7.4.1 割込みアクノリッジ・バス・サイクル」を参照してください。

ベクタ番号を得ると、プロセッサは例外ベクタ・オフセット、プログラム・カウンタ、およびステータス・レジスタの内部コピーをスーパバイザ・スタックにセーブします。セーブされるプログ

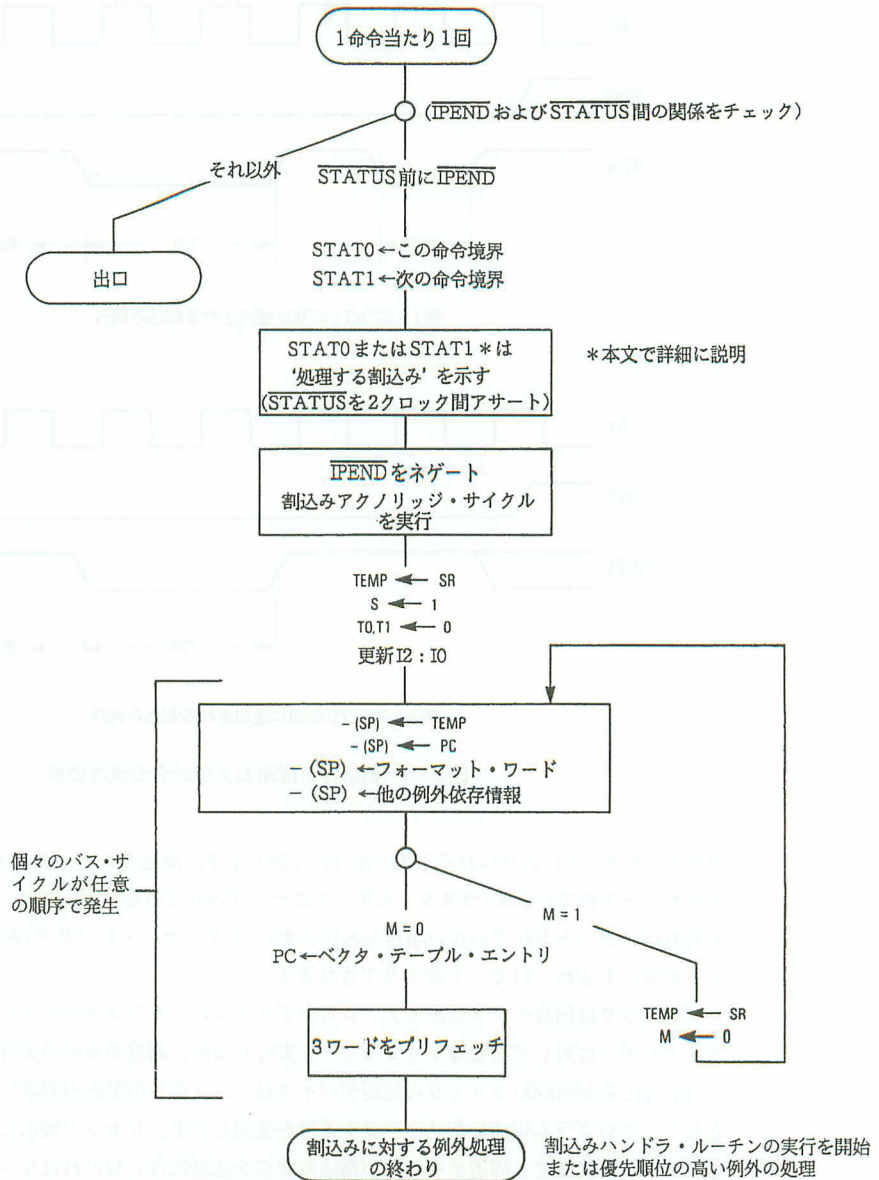



図8-5　割込み例外処理のフローチャート

ラム・カウンタの値は、割込みが発生しなければ実行するはずであった命令の論理アドレスです。コプロセッサ命令の実行中に割込みが検出された場合は、さらに詳しい内部情報がスタックにセーブされるため、割込みハンドラが実行を終了すると、MC68030がそのコプロセッサ命令を継続して実行することができます。

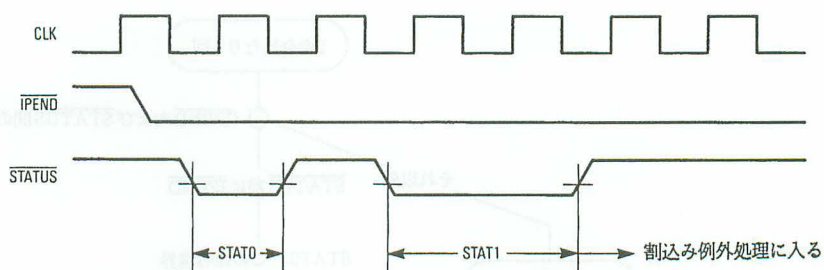
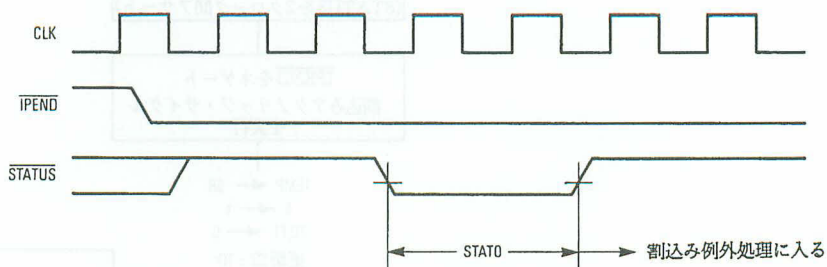
ステータス・レジスタのMビットがセットされている場合、プロセッサはMビットをクリアし、スローアウェイ(throwaway)例外スタック・フレームを割込みスタックの先頭に作成します。この第2のフレームには、マスタ・スタックの先頭に作成されるフレームと同じプログラム・カウンタおよびベクタ・オフセットが格納されますが、フォーマット番号は0または9ではなく1になっています。スローアウェイ・フレームにセーブされるステータ・レジスタのコピーは、Sビットを除いては

例1：STAT1の間に通知される割込み例外

例2：STAT0の間に通知される割込み例外

図8-6　割込みの認識および命令の境界の例

マスタ・スタックに入れられるものとまったく同じです。割込みスタックに入れられるものはSビットがセットされています(マスタ・スタックにセーブされているコピーでは、セットされていることもあれば、セットされていない場合もあります)。ステータス・レジスタ(例外の処理後)では、Sビットがセットされ、Mビットがクリアされます。

プロセッサは例外ベクタにあるアドレスをプログラム・カウンタにロードします。割込みハンドラ・ルーチンに対して必要なプリフェッチを実行した後、通常の命令の実行が再開されます。

ほとんどのM68000ファミリの周辺デバイスは、システムの割込み要求アクノリッジ機構の一部として、プログラム可能な割込みベクタ番号を使用します。リセット後にこのベクタ番号が初期化されていない状態で、周辺デバイスが割込み要求を認識応答しなければならない場合、通常周辺デバイスは未初期化(uninitialized)割込みベクタ、15を返します。

8. 1. 10　MMU構成例外

MC68030が無効なデータをMMUのTC、CRP、またはSRPレジスタに転送しようとするPMOVE命令を実行すると、PMOVE命令によってMMU構成例外が発生します。この例外は、ポスト命令例外で、命令の実行を終了してから処理されます。プロセッサはMMU構成例外が発生すると、例外ベクタ番号56を生成します。MMUレジスタの有効な構成については「第9章　メモリ管理ユニット」を参照してください。

プロセッサはステータス・レジスタをコピーし、スーパバイザ特権レベルに入り、トレース・ビットをクリアします。プロセッサはベクタ・オフセット、走査PC値(次の命令を指す)、およびステータス・レジスタのコピーをスーパバイザ・スタックにセーブします。また、スタックにあるPMOVE命令の論理アドレスもセーブします。最後に、プロセッサは例外ベクタのアドレスから必要なプリフェッチを行なった後、通常の命令実行を再開します。

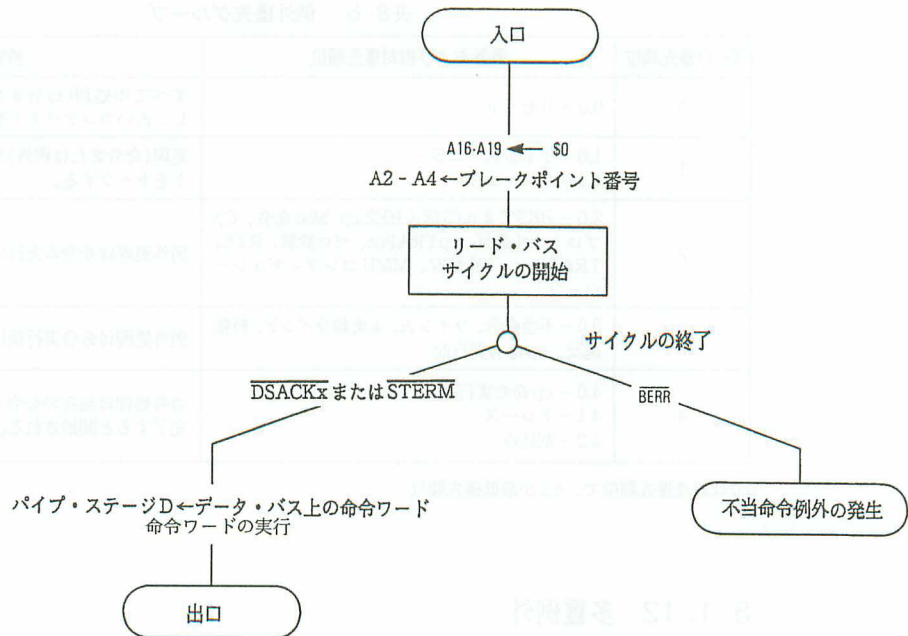



図8-7　ブレークポイント命令のフローチャート

8. 1. 11　ブレークポイント命令例外

MC68030をハードウェア・エミュレータで使用するには、エミュレータ・コードにブレークポイントを挿入し、各ブレークポイントで所定の動作を行なう手段が用意されていなければなりません。MC68000とMC68008では、ブレークポイントに不当命令を挿入しておき、そのベクタ・ロケーションから不当命令例外を検出することによってこれを行なうことができます。しかし、MC68010、MC68020、およびMC68030では、ベクタ・ベース・レジスタを任意にリロケートできるため、例外アドレスで確実にブレークポイントを識別することはできません。MC68020およびMC68030のプロセッサは、1組の不当命令($4848～$484F)を8つの固有のブレークポイントに対応させることによって、ブレークポイント機能を提供しています。このブレークポイント機能によって、大きな性能低下を招くことなく、オンチップ・キャッシュに存在するプログラムの実行を外部ハードウェアでモニタすることができます。

MC68030はブレークポイント命令を実行するときには、アドレス・ラインA2-A4をブレークポイント番号に対応させ、CPU空間$0からブレークポイント・アクノリッジ・サイクル(リード・サイクル)を実行します。CPU空間$0のアドレスについては図7-44、そしてブレークポイント・アクノリッジ・サイクルについては、「7. 4. 2　ブレークポイント・アクノリッジ・サイクル」を参照してください。

外部ハードウェアは命令ワードをデータ・バスに置いて、BERR、DSACKx、またはSTERMのいずれかを返すことができます。このバス・サイクルがBERRによって終了した場合、プロセッサは不当命令例外処理を実行します。また、このバス・サイクルがDSACKxまたはSTERMで終了した場合、プロセッサは返されたデータを使用して内部命令パイプにあるブレークポイント命令を置き換えて、その命令の実行を開始します。パイプの残りは変更されず、命令の実行にスタッキングやベクタのフェッチングは伴いません。図8-7にブレークポイント命令のフローチャートを示します。

表8-6　例外優先グループ

ｸﾞﾙｰﾌﾟ/優先順位	例外および相対優先順位	特性
0	0.0 − リセット	すべての処理(命令または例外)をアボートし、古いコンテクストをセーブしない。
1	1.0 − アドレス・エラー 1.1 − バス・エラー	処理(命令または例外)を中断し、コンテクストをセーブする。
2	2.0 − BKPT # n,CHK,CHK2,cp Mid 命令、Cp プロトコル違反、cpTRAPcc、ゼロ除算、RTE、TRAP # n、TRAPV、MMU コンフィギュレーション	例外処理は命令の実行の一部
3	3.0 − 不当命令、ラインA、未実装ラインF、特権違反、cp 命令実行前	例外処理は命令実行前に開始される。
4	4.0 − cp 命令実行後 4.1 − トレース 4.2 − 割込み	命令処理は現在の命令または前の例外処理が完了すると開始される。

0.0 は最高優先順位で、4.2 が最低優先順位

8. 1. 12　多重例外

複数の例外が同時に発生した場合は、決まった優先順位に従って処理されます。表8-6に特性によって分類した例外の一覧を示します。各グループには、1〜4の優先順位が付けられており、0が最も高い優先順位です。

MC68030がある条件に対する例外処理を完了したときに、別の例外が保留されていたときには、もとの例外に対する例外ハンドラを実行しないで、直ちに保留例外の例外処理を開始します。また、バス・エラーまたはアドレス・エラーが発生すると、その例外処理は優先順位の低い例外に優先し、すぐに実行されます。たとえば、トレース例外を処理している間にバス・エラーが発生した場合、システムはトレース例外の処理を完了する前に、そのバス・エラーを処理しそのハンドラを実行します。しかし、ほとんどの例外は、例外処理中には発生しません。また、表8-6に示す例外の組合せが同時に保留される可能性はほとんどありません。

この優先順位の決定方式は、複数の例外が同時に発生したときに、例外ハンドラを実行する順序を決める上できわめて重要です。

一般的なルールとして、例外の優先順位が低いほど、その例外のハンドラ・ルーチンが早く実行されます。たとえば、トラップ、トレース、および割込みの例外が同時に保留されている場合、最初にトラップ例外が処理され、その直後にトレースの例外処理が実行され、最後に割込みが処理されます。プロセッサが通常の命令の実行を再開したときにはプロセッサは割込みハンドラの中にあり、割込みハンドラの実行が終了すると、トレース・ハンドラに戻り、ついでトラップ例外ハンドラに戻ります。このルールが当てはまらないのはリセット例外で、リセット例外のハンドラは、優先順位が最高であっても最初に実行されます。というのは、リセットによって他の例外をすべてクリアしてしまうためです。

8. 1. 13　例外からの戻り

保留されているすべての例外に対する例外処理を終了したあと、プロセッサは最後に処理した例外のベクタにあるアドレスから通常の命令の実行を再開します。例外ハンドラが実行を終了したら、プロセッサは例外発生前のシステム・コンテクストに戻らなければなりません(可能であれば)。RTE 命令はどの例外でも、ハンドラから前のシステム・コンテクストに戻ります。

プロセッサはRTE 命令を実行すると、アクティブ・スーパバイザ・スタックの先頭にあるスタッ

ク・フレームをチェックして、それが有効なフレームかどうか、そしてどのタイプのコンテクストの回復が必要であるかを調べます。この章では、各スタック・フレームのタイプに対する処理を説明します。スタック・フレーム・タイプのフォーマットについては、「8.3　コプロセッサの検討事項」を参照してください。

通常の4ワード・フレームの場合、プロセッサはスタックから取り出したデータでステータス・レジスタおよびプログラム・カウンタを更新し、スタック・ポインタを8インクリメントしてから通常の命令の実行を再開します。

スローアウェイ(throwaway)型の4ワード・スタックの場合、図8-8に示すように、プロセッサはフレームからステータス・レジスタ(SR)値を読み出し、アクティブ・スタック・ポインタを8インクリメントし、スタックから読み出した値でステータス・レジスタを更新してから、再びRTEの処理を開始します。プロセッサはアクティブ・スタック先頭にあるスタック・フレーム(前の操作で使用したものと同じ場合もある)から新しいフォーマット・ワードを読み出し、そのフォーマットに対応した正しい操作を実行します。ほとんどの場合、スローアウェイ型のフレームは割込みスタックにあり、スタックからSR値を読み出したときには、SビットとMビットがセットされます。その場合、マスタ・スタックには通常の4ワード・フレームまたは10ワードのコプロセッサ用中間命令フレームがあります。しかし、第2フレームはどのフォーマットでもよく(別のスローアウェイ・フレームでもよい)、3つのシステム・スタックのどこにでも常駐可能です。

6ワードのスタック・フレームの場合、プロセッサはスタックからステータス・レジスタとプログラム・カウンタ値を回復し、アクティブ・スーパバイザ・スタック・ポインタを12インクリメント

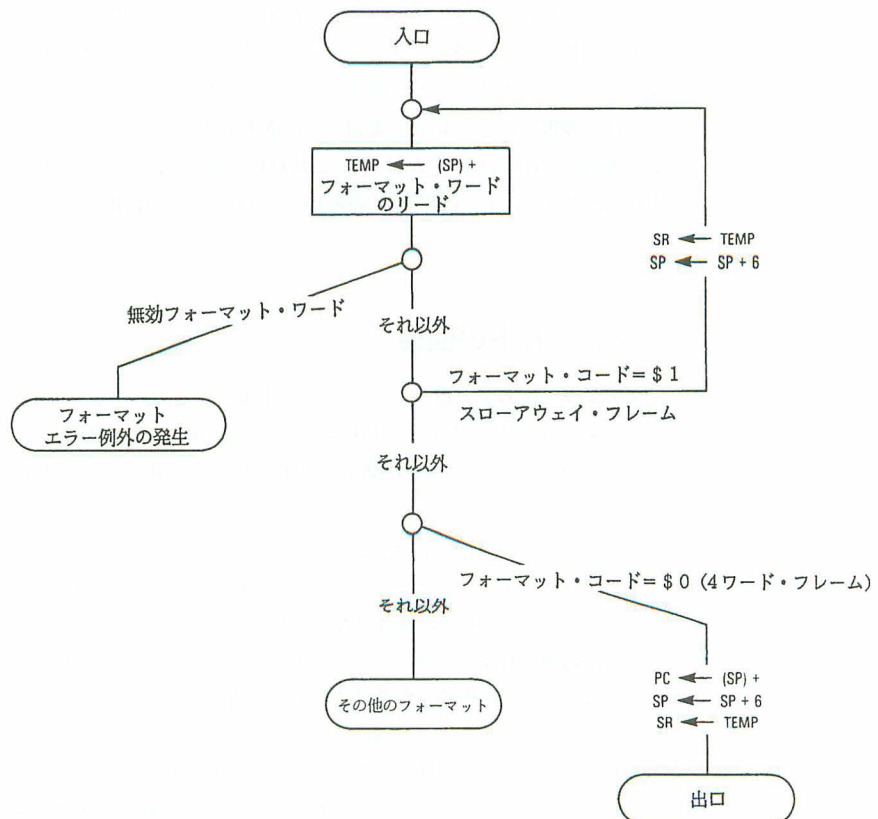



図8-8　スローアウェイ型4ワード・フレームのRTE命令

し、通常の命令の実行を再開します。

コプロセッサ用中間命令スタック・フレームの場合、プロセッサはステータス・レジスタ、プログラム・カウンタ、命令アドレス、内部レジスタ値、そして計算した実効アドレスをスタックから取り出して、対応する内部レジスタにロードしてから、スタック・ポインタを20インクリメントします。さらに、プロセッサはその例外を発生したコプロセッサの応答レジスタを読み出して、次に実行する操作を決めます。コプロセッサ関連の例外の詳細については、「第10章　コプロセッサ・インタフェースの説明」を参照してください。

ショートおよびロングのバス・フォールト・スタック・フレームの場合、プロセッサは最初にスタックのフォーマット値の妥当性をチェックします。さらに、ロング・スタック・フレームの場合、プロセッサはスタックにあるバージョン番号と自分のバージョン番号を比較します。バージョン番号は、ロング・スタック・フレームのロケーションSP＋＄36(16進)にあるワードの最上位ニブル(ビット12～15)にあります。この妥当性のチェックは、マルチプロセッサ・システムで、RTE命令がデータを正しく解釈するために必要です。RTE命令はスタック・フレームの両端から読出しを行なって、それがアクセス可能かどうか確認します。そのフレームが無効であったり、アクセスできないことが分かった場合、プロセッサはそれぞれフォーマット・エラーまたはバス・エラー例外を発生します。フレームが有効であった場合、プロセッサはそのフレーム全体を適切な内部レジスタに読み込んで、スタックの割当てを解除してから、通常の処理を再開します。プロセッサがフレームのロードを開始して内部状態を回復したら、$\overline{\text{BERR}}$信号によってホルト状態に入り、$\overline{\text{STATUS}}$信号を連続してアサートします。フレームを内部レジスタに読み込んだ後のプロセッサの状態に関する詳細は、「8. 2　バス・フォールトの回復」を参照してください。

上記エラーのいずれか、あるいは不当フォーマット・コードによって、RTE 命令の実行中にフォーマット・エラーまたはバス・エラーが発生した場合、プロセッサは使用しようとしていたフレームの下に、通常の4ワード・スタック・フレームまたはバス・サイクル・フォールト・スタック・フレームを作成します。このようにして、障害のあるスタック・フレームをそのままにしておきます。例外ハンドラは、障害のあるフレームを調べたり修復することができます。マルチプロセッサ・システムでは、障害のあるフレームは可能であれば、異なるタイプの別のプロセッサ(たとえばMC68010、MC68020、または将来のM68000プロセッサ)が使用できるよう、そのままにしておきます。

8. 2　バス・フォールトの回復

アドレス・エラー例外またはバス・エラー例外はバス・フォールトを示します。バス・エラーまたはアドレス・エラー時のプロセッサ状態のセーブについては、「8. 1. 2　バス・エラー例外」、そしてRTE命令によってプロセッサ状態を回復する方法は、「8. 1. 13　例外からの戻り」に説明してあります。

プロセッサがデータ・アイテムまたは命令ストリームのいずれかにアクセスするとバス・エラーが発生します。データ・アイテムにアクセスしているときに、バス・エラー例外が発生すると、バス・サイクルが終了するとすぐに例外処理に入ります。オンチップMMUからレポートされたバス・エラーもすぐに処理されます。命令ストリームへのアクセス中にバス・エラーが発生した場合は、プロセッサがそのアクセスで供給されるはずであった情報(もし、あれば)を利用しようとするときまで処理されません。命令フォールトの場合、ショート・フォーマット・フレームが適用されるときには、パイプ・ステージBワードはプログラム・カウンタの値＋4で、パイプ・ステージCワードはプログラム・カウンタの値＋2です。ロング・フォーマットの場合、SP＋＄24にあるロング・ワードはステージBワードのアドレスを保持しており、ステージCワードのアドレスはステージBワード

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FC	FB	RC	RB	X	X	X	DF	RM	RW	SIZE		X	FC2-FC0		

FC　　　－命令パイプのステージCでのフォールト
FB　　　－命令パイプのステージBでのフォールト
RC　　　－命令パイプのステージCに対する再実行フラグ＊
RB　　　－命令パイプのステージBに対する再実行フラグ＊
DF　　　－データ・サイクルに対するフォールト/再実行フラグ＊
RM　　　－データ・サイクルでのリード・モディファイ・ライト
RW　　　－データ・サイクルに対するリード/ライト　1＝リード、0＝ライト
SIZE　　－データ・サイクルに対するサイズ・コード
FC0～FC2－データ・サイクルに対するアドレス空間
＊1＝フォールトが発生したバス・サイクルの再実行、または保留プリフェッチの実行
0＝バス・サイクルを再実行しない
X＝内部使用のみ

図8-9　特殊ステータス・ワード(SSW)

のアドレス－2です。アドレス・エラー・フォールトは、命令ストリームへのアクセスにおいてのみ発生し、例外はそのバス・サイクルが実行される前に処理されます。

8. 2. 1　特殊ステータス・ワード

バス・フォールト例外スタック・フレーム情報の一部としてセーブされるレジスタ群の1つに、内部特殊ステータス・ワード(図8-9参照)があります。このワードはショート・バス・サイクル・フォールト用フォーマットおよびロング・バス・サイクル・フォールト用フォーマットの両方とも、オフセット＄Aに格納されます。バス・サイクル・フォールト用スタック・フレーム・フォーマットについては、この章の終わりのところで詳しく説明します。

特殊ステータス・ワード(SSW)の情報は、そのフォールトが命令ストリーム、データ・ストリーム、あるいはその両方で発生したかどうかを示します。SSWの上位半分には、命令パイプのステージBおよびCのそれぞれに対応する2個のステータス・ビットがあります。フォールト・ビット(FBおよびFC)は、プロセッサがステージ(BまたはC)を使用しようとしたが、そのステージのプリフェッチでのバス・エラーによって無効とマークされているのが分かったことを示します。このフォールト・ビットは、バス・エラー・ハンドラがバス・エラー例外の原因を知るために使用することができます。対応するステージのプリフェッチ中にフォールトが発生すると、再実行フラグ・ビット(RBおよびRC)がセットされ、フォールトが発生したことを示します。再実行ビットは対応するフォールト・ビットがセットされているときには常にセットされます。再実行ビットは、命令パイプのステージのワードが無効であることを示し、ハンドラはこのビットの状態を使用し、必要に応じてアドレス・エラーまたはバス・エラー後のパイプの値を修復することができます。プロセッサがRTE命令を実行するときに再実行ビットがセットされていた場合、プロセッサはバス・サイクルを実行してパイプの対応するステージのための命令ワードをプリフェッチします(必要な場合)。パイプのステージに対して再実行およびフォールト・ビットがセットされていた場合、RTE命令は自動的にそのステージに対するプリフェッチ・サイクルを再実行します。バス・サイクルのアドレス空間は、スタックにあるステータス・レジスタのコピーで指示される特権レベルに対するプログラム空間です。再実行ビットがクリアされている場合、パイプの対応するステージのためのスタックにあるワードは有効として受け入れられ、プロセッサは対応するステージに保留されているプリフェッチがなく、また必要に応じてソフトウェアがそのステージのイメージを修復したか充てんしたとみなします。

アドレス・エラー例外が発生した場合、スタック・フレームに書き込まれるフォールト・ビット

はセットされません(フォールト・ビットは、前述したようにバス・エラーによってのみセットされる)。したがって、再実行ビットしか例外の原因を示すものはありません。パイプラインの状態によって、RBとRCが両方ともセットされるか、あるいはRBだけがセットされます。パイプラインを修復して中断されていた命令の実行を継続する必要がある場合、再実行ビットの状態に基づいて、ステージCまたはステージB、あるいはその両方のイメージの中にソフトウェアによって正しい命令ストリーム・データを置き、その後対応する再実行ビットをクリアしなければなりません。SSWの下位半分はデータ・サイクルにしか適用されません。SSWのDFビットがセットされている場合、データ・フォールトが発生し、例外となっています。プロセッサがスタック・フレームから読出しを行なったときにDFビットがセットされていた場合、プロセッサはそのフォールトが発生したデータ・アクセスを再実行します。DFビットがセットされていなっかた場合は、読出しのときにスタックのデータ入力バッファが有効であったか、書込みのときにデータが正しくメモリに書き込まれた(あるいは、データ・フォールトが発生しなかった)ことを示します。SSWのRMビットはリード・モディファイ・ライト操作を示し、RWビットはそのサイクルがリード操作であったかライト操作であったかを示します。SIZフィールドはオペランド・アクセスのサイズを示し、FCフィールドはデータ・サイクルのアドレス空間を指定します。データおよび命令ストリームのフォールトは、同時に保留されることがあるため、フォールト・ハンドラはFC、FB、RC、RB およびDFの各ビットの任意の組合せを認識できなければなりません。

8. 2. 2 ソフトウェアによるバス・サイクルの終了

フォールトが発生したバス・サイクルを終了させる方法の1つは、ソフトウェア・ハンドラを使用してそのサイクルをエミュレートすることです。アドレス・エラーを修正するのはこの方法しかありません。ハンドラは、フォールトを発生した命令に対してトランスペアレントになるように、フォールトが発生したバス・サイクルをエミュレートしなければなりません。命令ストリーム・フォールトの場合、ハンドラは命令パイプのステージBおよびCの両方に対してバス・サイクルを実行する必要があります。RBおよびRCビットはバス・サイクルを必要とするステージを必要とします。FBおよびFCビットはステージの内容を使用しようとしたときに、それが無効であったことを示します。これらのステージは修復しなければなりません。フォールトが発生した各ステージに対して、ソフトウェア・ハンドラは、スタックにセーブされているステータス・レジスタのコピーのSビットで示されている正しいアドレス空間から、命令ワードをフェッチして、それをスタック・フレームの適切なステージのイメージに書き込む必要があります。さらに、ハンドラは修復したステージに関連する再実行ビットをクリアしなければなりません。ハンドラはフォールト・ビットFBおよびFCの値を変更してはなりません。

データ・フォールト(DF＝1で示す)を修復するために、ソフトウェアは最初にSSWのRMビットを調べて、リード・モディファイ・ライト・サイクル中にフォールトが発生したかどうかを確認しなければなりません。RM＝0の場合、ハンドラはSSWのR/Wビットをチェックしてそのフォールトがリードまたはライト・サイクルで生じたものかどうかを確認しなければなりません。データ・ライト・フォールトの場合、ハンドラはスタック・フレームのデータ出力バッファ(DOB)から、正しいサイズのデータを、SSWで指定されるアドレス空間のデータ・フォールト・アドレスで示されるロケーションへ転送しなければなりません。(DOBとデータ・フォールト・アドレスの両方とも、スタック・フレームの一部であり、それぞれSP＋＄18およびSP＋＄10にあります)。データ・リード・フォールトは、ロング・バス・フォールト・フレームしか発生せず、ハンドラはフォールト・アドレスおよびアドレス空間によって示されているロケーションから、正しいサイズのデータをロング・フォーマットのスタック・フレームのロケーションSP＋＄2Cにあるデータ入力バッファ(DIB)のイメージへ転送しなければなりません。バイト、ワード、および3バイトのオペランドが、4バイ

トのデータ・バッファに右詰めされます。さらに、ソフトウェア・ハンドラはSSW のDFビットをクリアして、フォールトしたデータ・バス・サイクルが修正されたことを知らせなければなりません。

リード・モディファイ・ライト・サイクルをエミュレートするために、例外ハンドラはまずプログラム・カウンタ・アドレス(スタック・フレームのSP＋2)にあるオペレーション・ワードを読み込みます。このワードはCAS 、CAS2、またはTAS のうちどの命令でフォールトが発生したかを識別します。次に、ハンドラはこの命令全体(最高4ロング・ワード転送で構成される)をエミュレートして、ステータス・レジスタのコンディション・コード部分も適宜更新しなければなりません。なぜなら、RTE命令は、RMビット(SSW)がセットされ、かつDFビット(SSW)がクリアされている場合、全体の操作が完了しているものと予測しているためです。

命令全体をエミュレートするには、ハンドラは命令のデータ・レジスタおよびアドレス・レジスタをセーブしなければなりません(たとえば、MOVEM命令により)。次にハンドラは、メモリ・ロケーションの読出しと変更を(必要があれば)行ないます。ハンドラはスタック・フレームのSSWのDFビットをクリアし、ステータス・レジスタ・コピーのコンディション・コード、およびCASおよびCAS2命令に必要なデータまたはアドレス・レジスタのコピーを変更します。最後に、ハンドラはエミュレーションの初めにセーブしたレジスタを回復します。データ入力バッファ(DIB)、ステータス・レジスタのコピー、および特殊ステータス・ワードを除いて、ハンドラはバス・フォールト・スタック・フレームを変更してはなりません。特殊ステータス・ワードで変更可能なビットは、DF、RB、およびRCだけです。内部で使用するものも含めて、他のビットはすべて変更してはなりません。

アドレス・エラーはソフトウェアで修復しなければなりません。アドレス・エラー・フォールトは、フォーマット・ワードのベクタ・オフセット・フィールドによって、バス・エラー・フォールトと区別することができます。

8.2.3 RTE命令によるバス・サイクルの終了

フォールトの発生したバス・サイクルを終了させるもう1つの方法は、例外ハンドラを終了するRTE命令の実行中に、プロセッサにそのバス・サイクルを再実行させることです。アドレス・エラーから回復するのにこの方法を使用してはなりません。RET命令は常に実行されます。ハンドラ・ルーチンがエラーを修正し、フォールトをクリアした(そして、特殊ステータス・ワードの再実行およびDFビットをクリアした)場合、RTE 命令を実行することによって、バス・サイクルを終了させることができます。RTE命令の実行時にDFビットがセットされたままになっている場合は、RTE命令によってフォールトの発生したデータ・サイクルが再実行されます。パイプのステージのフォールト・ビットがセットされ、対応する再実行ビットがソフトウェアによってクリアされなかった場合は、RTE命令は関連する命令プリフェッチを再実行します。仮想メモリ・システムでページが存在しなかった場合など、そのフォールトの原因が修正されなかった場合は、再びフォールトが発生します。パイプのステージに対して再実行ビットがセットされ、フォールト・ビットがクリアされた場合は、関連のプリフェッチ・サイクルがRTE命令で実行されることもあり、されないこともあります(そのステージが必要か否かによる)。

RTE命令がそのバス・サイクルを再実行しようとしたときにフォールトが発生した場合は、プロセッサは前のフレームの割当て解除を行なった後、スーパバイザ・スタックに新しいスタック・フレームを生成し、アドレス・エラーまたはバス・エラー例外処理が通常どおり開始されます。

MC68030のリード・モディファイ・ライト操作も、ハンドラ・ルーチンを終了するRTE命令によって終了させることができます。SSWのDFビットをセットしたRTE命令によって実行される再実行操作は、命令全体を再実行します。エラーの原因が修正された場合は、ハンドラは命令をエミ

ュレートする必要はありませんが、DFビットをセットしたままにして、RTE命令を実行することができます。

システム・プログラマおよびシステム設計者は、MC68030のメモリ管理ユニットがR/W 信号の状態に関係なく、$\overline{RMC}$信号がアサートされているすべてのバス・サイクルを保護チェックの対象となるライト操作として扱うことを銘記しておいてください。そうしなかった場合、ライト操作の一部が行なわれ、残りがバス・エラーによってアボートされ、CAS 命令およびCAS2命令でシステム・ポインタを部分的に破壊してしまう可能性があります。

8. 3　コプロセッサの検討事項

例外ハンドラ・プログラムを書くときには、コプロセッサ命令実行中に発生するおそれのある例外(たとえば、バス・エラー、割込み、およびコプロセッサ関連例外)に対するハンドラ・ルーチンの始めと終わりにコプロセッサのコンテクストをセーブおよびリストアするか否かを慎重に検討しなければなりません。コプロセッサおよび例外ハンドラ・ルーチンの特質によって、cpSAVEおよびcpRESTORE命令によって、1つまたはそれ以上のコプロセッサの状態をセーブするかどうかが決まります。コプロセッサが複数のコプロセッサ命令を同時に実行することを許している場合は、ハンドラ・ルーチン中にコプロセッサがアクセスされるか否かに関係なく、コプロセッサが生成したすべての例外に対して、その状態をセーブし、リストアしなければならない場合があります。MC68882浮動小数点コプロセッサがこの種のコプロセッサ例です。他方、MC68881浮動小数点コプロセッサは、例外ハンドラ自身がコプロセッサを使用する場合にのみ、例外ハンドラ・ルーチン内でFSAVEまたはFRESTORE命令を必要とします。

8. 4　例外スタック・フレーム・フォーマット

MC68030は例外処理のために6種類のスタック・フレームを提供します。すなわち、通常の4ワードおよび6ワードのスタック・フレーム、4ワードのスローアウェイ型スタック・フレーム、コプロセッサの「命令途中での例外(mid-instruction exception)」スタック・フレーム、そしてショートおよびロング・バス・フォールト用スタック・フレームを発生します。

MC68030がスタック・フレームのリード/ライトを行なうときは、可能なかぎりロング・ワード・オペランド転送を使用します。したがって、32ビット・ポートにメモリがあり、スタック・ポインタがロング・ワードの境界に揃っている場合、例外処理の性能が大幅に向上します。また、プロセッサは必ずしもスタック・フレーム・データを順次読み書きするとはかぎりません。

システム・ソフトウェアは、特定の例外が特定のスタック・フレームを生成するように限定すべきではありません。将来のデバイスとの互換性を維持するため、ソフトウェアはどのタイプの例外についても、あらゆる種類のスタック・フレームが扱えるようになっていなければなりません。

表8-7にM68000ファミリに対して定義されているスタック・フレームを要約します。

表8-7　例外スタック・フレーム(その1)

スタック・フレーム	例外のタイプ(スタックされたPCが指すもの)
15　0 SP → ステータス・レジスタ +$02 プログラム・カウンタ +$06 0 0 0 0 ベクタ・オフセット 4ワード・スタック・フレーム――フォーマット$0	●割込み　[次の命令] ●ﾌｫｰﾏｯﾄ･ｴﾗｰ　[RTEまたはcpRESTORE命令] ●TRAP＃N　[次の命令] ●不当命令　[不当命令] ●A系列命令　[A系列命令] ●F系列命令　[F系列命令] ●特権違反　[特権違反の原因となっている命令の第1ワード] ●コプロセッサ命令実行前 [命令実行前での例外処理要求(Take Pre-Instruction)プリミティブを返した命令のオペ・ワード]
15　0 SP → ステータス・レジスタ +$02 プログラム・カウンタ +$06 0 0 0 1 ベクタ・オフセット スローアウェイ4ワード・スタック・フレーム――フォーマット$1	●割込み例外の処理中にマスタ状態から割込み状態への遷移が発生したときに、割込みスタックで生成される。 [次の命令――マスタ・スタックと同じ]
15　0 SP → ステータス・レジスタ +$02 プログラム・カウンタ +$06 0 0 1 0 ベクタ・オフセット +$08 命令アドレス 6ワード・スタック・フレーム――フォーマット$2	●CHK ●CHK2 ●cpTRAPcc ●TRAPcc ●TRAPV ●Trace ●ゼロ除算 ●MMUコンフィギュレーション ●コプロセッサ命令実行後 [これらすべての例外の次の命令] 命令アドレスは例外を引き起こした命令のアドレス
15　0 SP → ステータス・レジスタ +$02 プログラム・カウンタ +$06 1 0 0 1 ベクタ・オフセット +$08 命令アドレス +$0C 内部レジスタ、4ワード +$12 ｺﾌﾟﾛｾｯｻ命令途中例外ｽﾀｯｸ･ﾌﾚｰﾑ(10ﾜｰﾄﾞ)―ﾌｫｰﾏｯﾄ$9	●コプロセッサ命令途中 ●メイン――検出プロトコル違反 ●コプロセッサ命令実行中の割込み検出(ヌル割込み許可状態での再来要求(null come again with interrupts allowed) プリミティブ) [これらのすべての例外に対して命令ストリームからフェッチする次のワード] 命令アドレスは例外を引き起こした命令のアドレス。

表8-7　例外スタック・フレーム(その2)

スタック・フレーム	例外のタイプ(スタックされたPCが指すもの)
ショート・バス・サイクル・フォールト・スタック・フレーム(16ワード) ──フォーマット$A	●アドレス・エラーまたはバス・エラー ──命令境界での実行ユニット　[次の命令]
ロング・バス・サイクル・フォールト・スタック・フレーム(46ワード) ──フォーマット$B	●アドレス・エラーまたはバス・エラー ──命令実行中 [フォールト発生時に実行中であった命令のアドレス──フォールトのバス・サイクルを引き起こした命令ではないこともある]

第 9 章
メモリ管理ユニット

MC68030はデマンド・ページ方式の仮想記憶環境をサポートする、メモリ管理ユニット(MMU)を内蔵しています。プログラムで必要なメモリ領域を前もって指定せず、論理アドレスにアクセスすることによって領域を要求するということから、このメモリ管理を“デマンド”方式といいます。物理メモリはページ化されます。これは、物理メモリがページ・フレームと呼ばれる同一サイズのブロックに分割されることを意味します。論理アドレス空間もそれと同じサイズのページに分割されます。オペレーティング・システムは、プログラムを満たすのに必要なページをページ・フレームに割り当てます。

MMUの主な機能は、メモリに記憶した変換テーブルを使用して、論理アドレスを物理アドレスに変換することです。MMUにはアドレス変換キャッシュ(ATC)があり、この中には最後に使用された論理アドレスから物理アドレスへの変換値が記憶されています。MMUはCPUコアから論理アドレスを受け取ると、ATCをサーチしてそれに対応する物理アドレスをさがします。この変換値がATCにない場合、プロセッサはメモリ内の変換テーブルをサーチしてこの変換値をさがします。このサーチに必要なアドレス計算とバス・サイクルは、MC68030にあるマイクロコードと専用ロジックで実行されます。さらに、MMUには2つのトランスペアレント変換レジスタ(TT0およびTT1)があり、これらはアドレス変換なしでアクセスできるメモリ・ブロックを識別します。MMUの特長は次のとおりです。

- 32ビットの論理アドレスを32ビットの物理アドレスに変換
- 物理アドレス空間への2クロック・サイクル・アクセスをサポート
- データおよび命令キャッシュへのアクセスと並行してアドレス変換を実行
- フル・アソシエイティブ22エントリのATCを内蔵
- マイクロコードによる変換テーブル・サーチの制御
- 8種類のページ・サイズ： 256、512、1K、2K、4K、8K、16K および 32K バイト
- ユーザおよびスーパバイザ変換テーブル・ツリーを別々にサポート
- 2つの独立したブロックをトランスペアレント(変換なし)に定義可能
- 複数レベルの変換テーブル
- 0-15の上位論理アドレス・ビットを無視するためのイニシャル・シフト機能
- テーブルの一部を未定義とするためのリミット機能
- 書込み保護およびスーパバイザ保護
- ページ・ディスクリプタ内でヒストリ・ビットを自動的に保持
- キャッシュ・インヒビット出力($\overline{\mathrm{CIOUT}}$)信号をページ単位にアサート
- MMU変換ディセーブル入力信号($\overline{\mathrm{MMUDIS}}$)
- MC68851で定義される命令セットのサブセット

ATC内に変換値がある場合、MMUはアドレス変換時間を他の処理動作と完全にオーバラップさせます。ATCのアクセスは、オンチップ命令やデータ・キャッシュと並行して動作します。

図9-1はMC68030のブロック図で、MMUと実行ユニットおよびバス・コントローラとの関係を示しています。命令またはオペランド・アクセスについては、MC68030はキャッシュをサーチすると同時にATC内の物理アドレスをサーチします。変換値が利用できる場合(ATCヒット)、MMUはバス・コントローラに物理アドレスを与え、バス・サイクルの継続が可能です。命令またはリードされるオペランド・データがオンチップ・キャッシュにあるときは、アドレス・ストローブがアサートされる前に、バス・コントローラがバス・サイクルをアボートします。同様に、ATCで有効な変換値が利用できない場合、または無効なアクセスが行なわれた場合、MMUはアドレス・ストローブがアサートされる前に、バス・サイクルをアボートします。

MMUディセーブル入力信号($\overline{\text{MMUDIS}}$)は、エミュレーション、診断、またはその他の目的でアドレス変換をダイナミックにディセーブルするためのものです。

MMUのプログラミング・モデル(図9-2)は、2つのルート・ポインタ・レジスタ、1つの制御レジスタ、2つのトランスペアレント変換レジスタ、および1つのステータス・レジスタで構成されています。これらのレジスタは、スーパバイザ・プログラムからしかアクセスできません。CPUルート・ポインタ・レジスタは、ユーザ・アクセス、またはスーパバイザ・アクセスのための、論理アドレスから物理アドレスへのマッピングを記述するメモリ中のアドレス変換ツリー構造を指しています。スーパバイザ・ルート・ポインタ・レジスタは、オプションによりスーパバイザ・マッピング用のアドレス変換ツリー構造を指します。変換制御レジスタは、変換動作を制御するフィールドで構成されています。各トランスペアレント変換レジスタは、物理アドレスとして(変換なしに)使用される論理アドレス・ブロックを定義することができます。MMUステータス・レジスタには、PTEST命令の一部として処理される変換作業により得られたステータス情報が保持されています。

MMUのアドレス変換キャッシュ(ATC)は、22個の論理アドレスから物理アドレスへの変換値、および関連のページ情報を記憶するフル・アソシエイティブ・キャッシュです。ATCは、プロセッサが内部で与える論理アドレスおよびファンクション・コードをATC内のすべてのタグ・エントリと比較します。アクセス・アドレスとファンクション・コードがATC内のタグと一致(ヒットする)し、かつアクセス違反が検出されなかった場合、ATCは対応する物理アドレスをバス・コントローラに出力し、バス・コントローラは外部バス・サイクルを続行します。ファンクション・コードはそのまま修正なしにバス・コントローラへ送られます。

各ATCエントリには、論理アドレス、物理アドレス、およびステータス・ビット群があります。ステータス・ビット群の中には、書込み保護ビットとキャッシュ・インヒビット・ビットがあります。

ATCに論理アドレスに対応した変換値がなく(ミスが発生)、外部バス・サイクルが必要な場合、MMUはそのアクセスをアボートし、プロセッサに正しい変換を行なうためにメモリ内の変換テーブルをサーチするバス・サイクルを開始させます。テーブル・サーチがエラーなしで終了した場合、MMUはATCに変換値を記憶しアクセスの物理アドレスを与えるため、バス・コントローラは最初のバス・サイクルを再試行することができます。

MMU変換テーブルには、ルート・ポインタ・ディスクリプタが定義した最初のテーブルをベースとするツリー構造があります。カレント変換テーブルのルート・ポインタ・ディスクリプタは、2つのルート・ポインタ・レジスタの1つに常駐しています。一般的なツリー構造を図9-3に示します。ツリーの上位レベルにあるテーブル・エントリは、別のテーブルを指しています。テーブル・リーフ・エントリはページ・フレーム・アドレスです。変換テーブルに記憶されているアドレスはすべて物理アドレスであり、変換テーブルは物理アドレス空間に存在します。

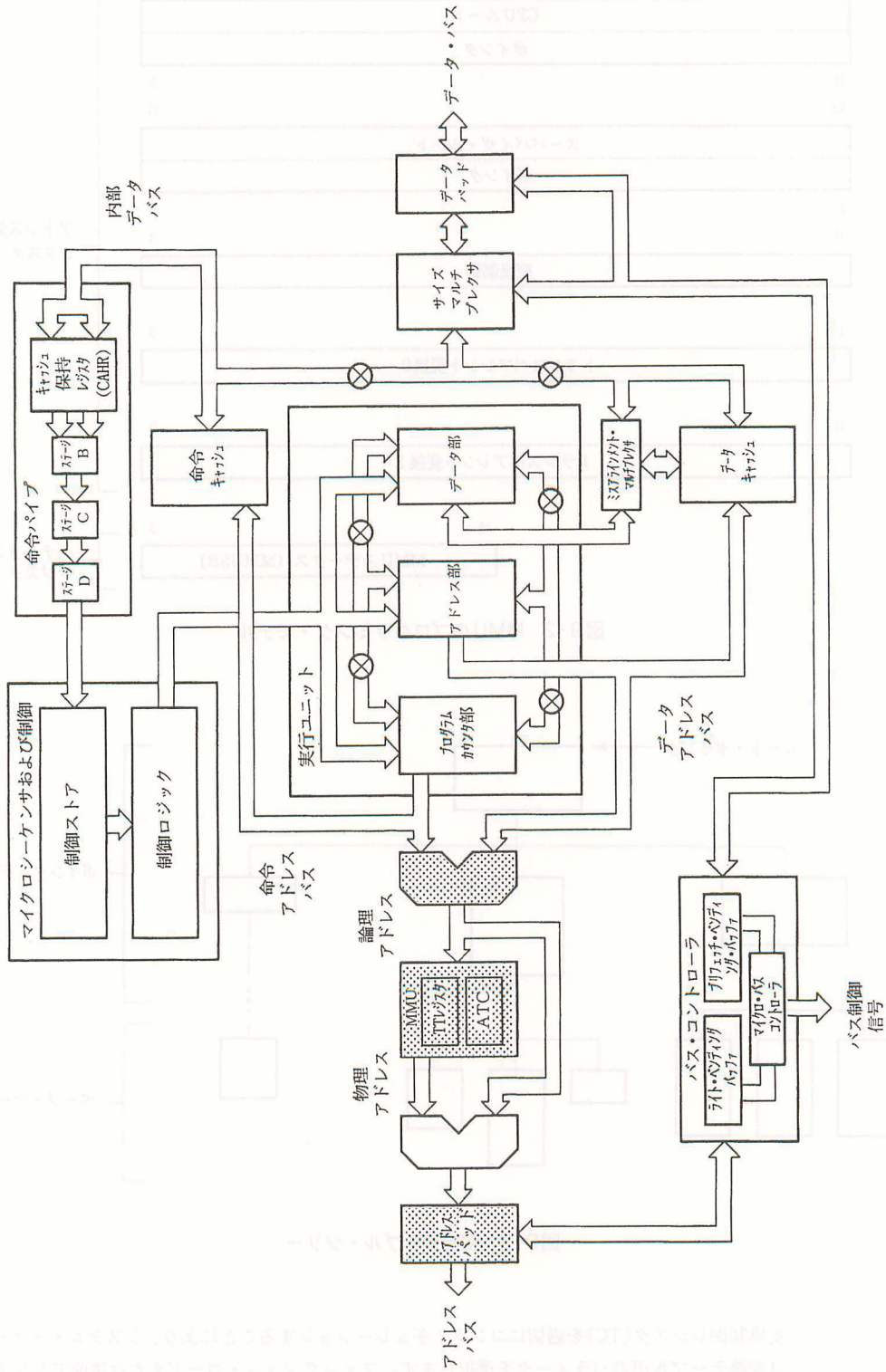



図9-1　MMUのブロック図

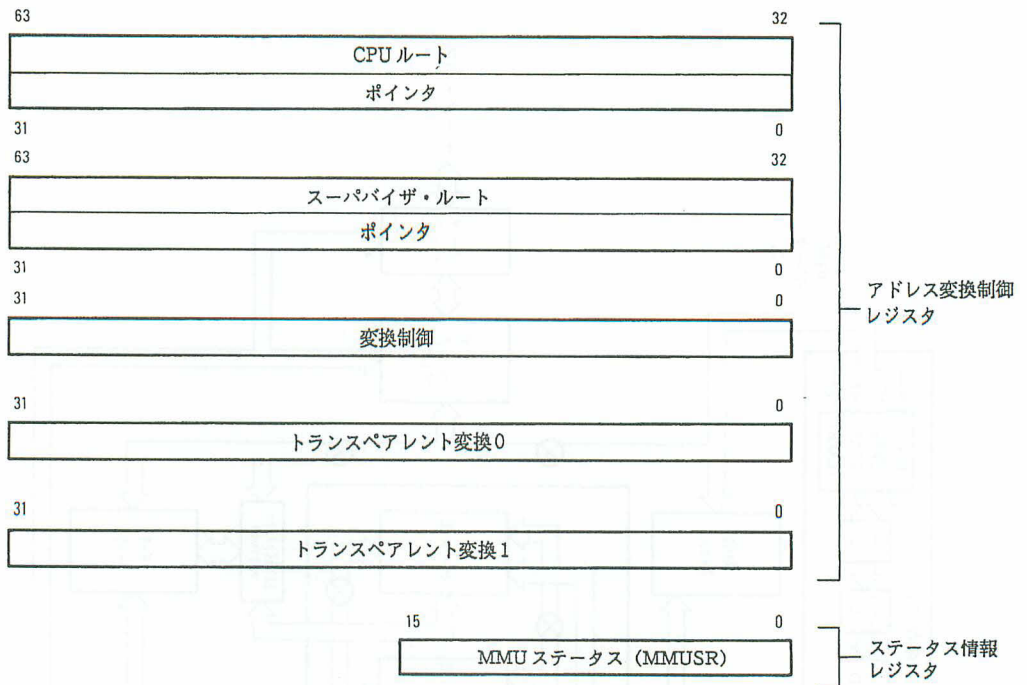



図9-2　MMUのプログラミング・モデル

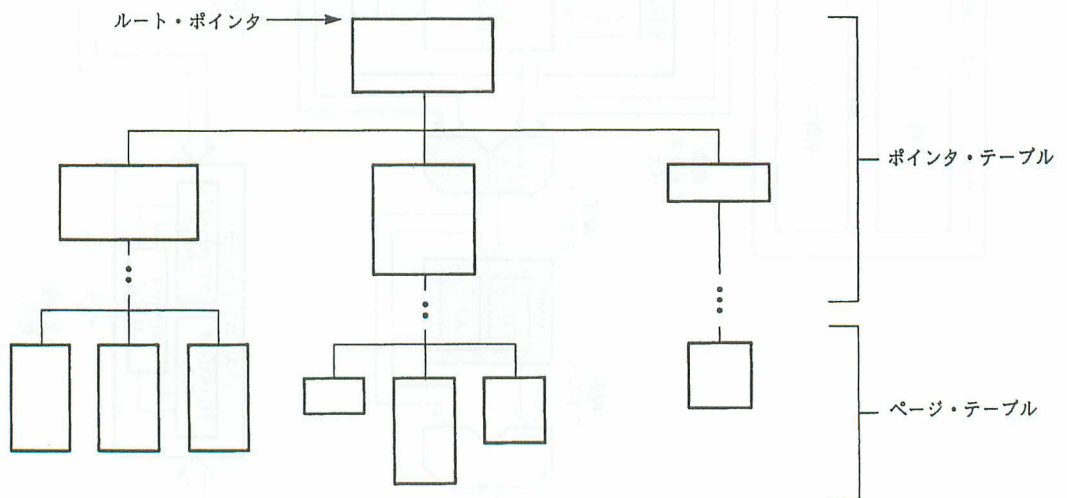



図9-3　変換テーブル・ツリー

変換制御レジスタ(TC)を適切にコンフィギュレーションすることにより、システム・ソフトウェアは変換テーブル用のパラメータを選択します。ファンクション・コードまたは論理アドレスの一部をテーブルの最初のルックアップ・レベルへのインデックスとして定義することができます。TCレジスタは、ルックアップの各レベルについて、論理アドレスの何ビットをインデックスとして使用するか(1つのレベルで15ビットまで使用可能)をTCレジスタが指定します。

9.1　変換テーブルの構造

MC68030はATCとメモリに記憶された変換テーブルを使用して論理アドレスから物理アドレスへの変換を行ないます。プログラム用の変換テーブルは、オペレーティング・システムがメモリにロードします。

MC68030がサポートする一般的な変換テーブル構造は、ツリー構造のテーブルです。ポインタ・テーブルはツリーの分岐を形成します。これらのテーブルには他のテーブルのベース・アドレスがあります。ページ・ディスクリプタは、ポインタ・テーブル内に常駐することができ、その場合はアーリ・ターミネーション・ディスクリプタといいます。ツリーの葉に当たるテーブルをページ・テーブルとよびます。いつの時点で見ても、メモリ内に常駐を要求されるのは、論理アドレス空間全体に対する変換テーブルのほんの一部分です。特に、現在実行中のプロセスが使用している論理アドレスを変換するテーブルの一部分だけが常駐を必要とするのです。プロセスが追加メモリを必要とするときには、変換テーブルの一部分をダイナミックに割り当てることが可能です。

図9-4に示すように、テーブルのルート・ポインタは、ツリーの最上位にあるテーブルのベース・アドレスを含むディスクリプタです。ポインタ・テーブルとページ・テーブルもディスクリプタで構成されます。ポインタ・テーブルのディスクリプタには、通常ツリーの次のレベルにあるテーブルのベース・アドレスがあります。テーブル・ディスクリプタは、次のテーブルへのインデックスのリミット、保護情報、およびそのディスクリプタに関するヒストリ情報をもつこともできます。各テーブルは、論理アドレスから抽出されたフィールドによってインデックスが付けられます。図9-4に示す例は、論理アドレス＄00AのAフィールドがルート・ポインタ値に加算され、変換ツリーのAレベルでのディスクリプタの選択に使用されます。選択されたディスクリプタは適切なページ・テーブルのベースを指し、論理アドレス(＄006)のBフィールドがこのベース・アドレスに加算され、ページ・テーブル内のディスクリプタの選択に使用されます。ページ・テーブルのディスクリプタには、ページの物理ベース・アドレス、保護情報、およびそのページのヒストリ情報があります。連続したページ・ブロックを定義するために、ページ・ディスクリプタをポインタ・テーブル内またはルート・ポインタ内に置くことも可能です。2レベルのページ・タスクを示します。32ビットの論理アドレス空間は、それぞれが1024バイトの4096個のセグメントに分割されています。

図9-5にメモリ内の変換ツリーのレイアウト例を示します。

9.1.1　変換制御

変換制御レジスタ(TC)は、メモリ内でのページ・サイズを定義し、スーパバイザ・アクセスに使用されるルート・ポインタ・レジスタを選択し、変換ツリーの最上位レベルにファンクション・コードでインデックスを付けるかどうかを指示し、さらにさまざまな変換ツリーのレベルにインデックスを付けるのに使用する論理アドレスのビット数を指定します。TCレジスタのイニシャル・シフト(IS)フィールドは、論理アドレス空間のサイズを定義し、変換テーブルのルックアップ時に無視する最上位アドレス・ビット数を保持しています。たとえば、ISフィールドが0にセットされていれば、論理アドレス空間は2^{32}バイトです。一方、ISフィールドが15にセットされていれば、論理アドレス空間には$2^{32}-2^{15}$バイトしか含まれません。

TCレジスタのページ・サイズ(PS)フィールドは、そのシステムに対するページ・サイズを指定します。システムのページ数は、論理アドレス空間をページ・サイズで割った値に等しくなります。変換ツリーで定義可能な最大ページ数は、1600万($2^{32}/2^{8}$)以上になります。最小値は4($2^{17}/2^{15}$)です。テーブル・ルックアップにファンクション・コードを使用して、上記のサイズの領域を7つ(FC＝0-6)まで定義することができます。各種サイズの変換テーブルによって、論理アドレス空間の全範囲を定義することができます。MC68030では大規模な変換テーブルを容易に実現できます。

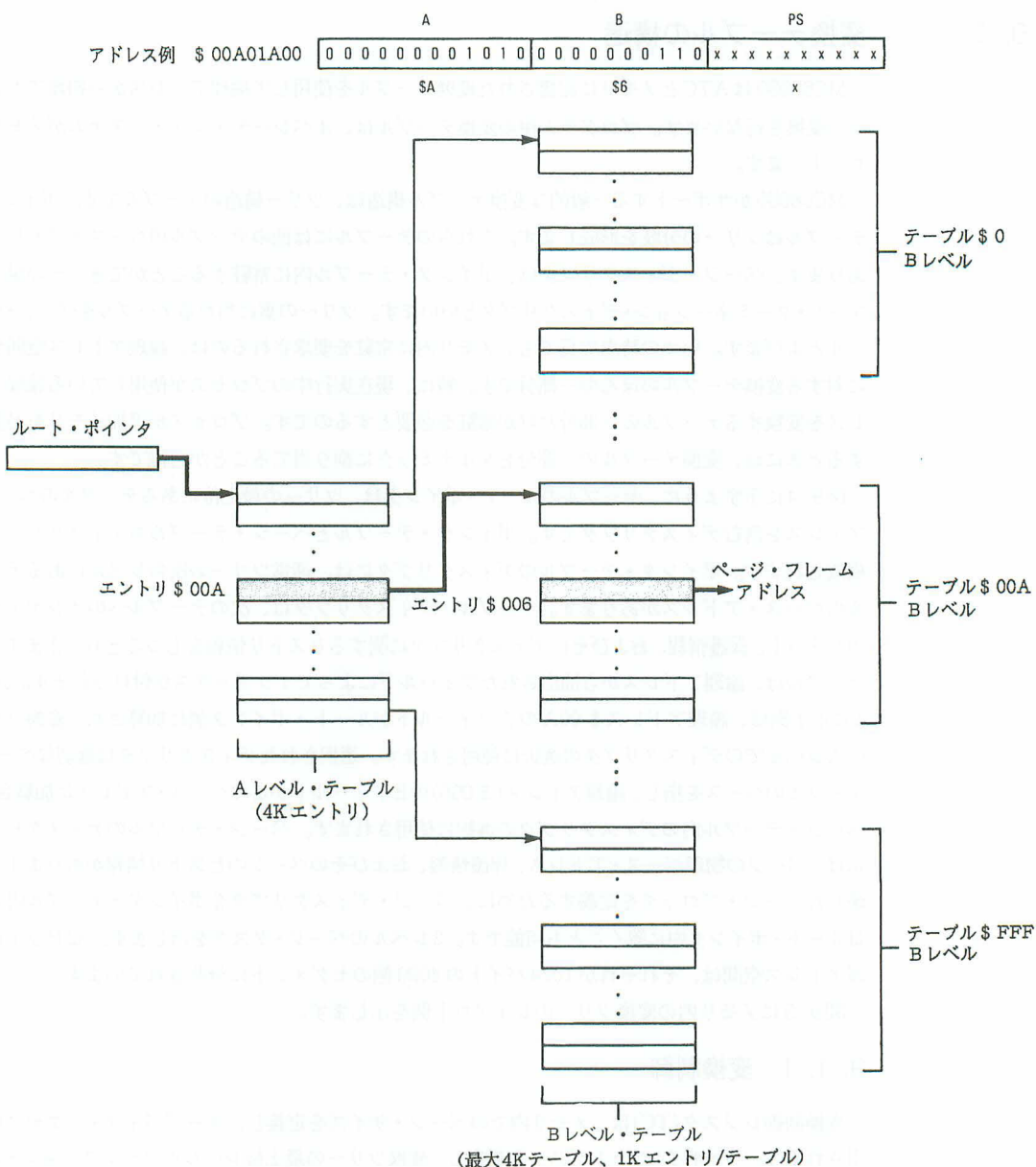



図9-4　変換テーブル・ツリーの例

5レベルのテーブルをもつツリー構造を使用することにより、変換テーブルの設計に融通性を与えることができます。ルート・ポインタのリミット・フィールドは、最初のインデックスの値を制限することができ、それによって実際に必要なディスクリプタ数を制限します。オプションにより、構造の最上位レベルにファンクション・コード・ビットでインデックスを付けることができます。この場合、このレベルのポインタ・テーブルは8個のディスクリプタを保持します。構造の次のレベル(TCレジスタのFCLビットがゼロにセットされているときには、最上位レベル)は、論理アドレスの最上位ビットでインデックスを付けます(ISフィールドで指定されるビット数を無視)。このインデックスに使用する論理アドレス・ビット数は、TCレジスタのTIAフィールドで指定されます。たとえば、TIAフィールドの値が5の場合、このレベルに対するインデックスには5ビットが含まれ、こ

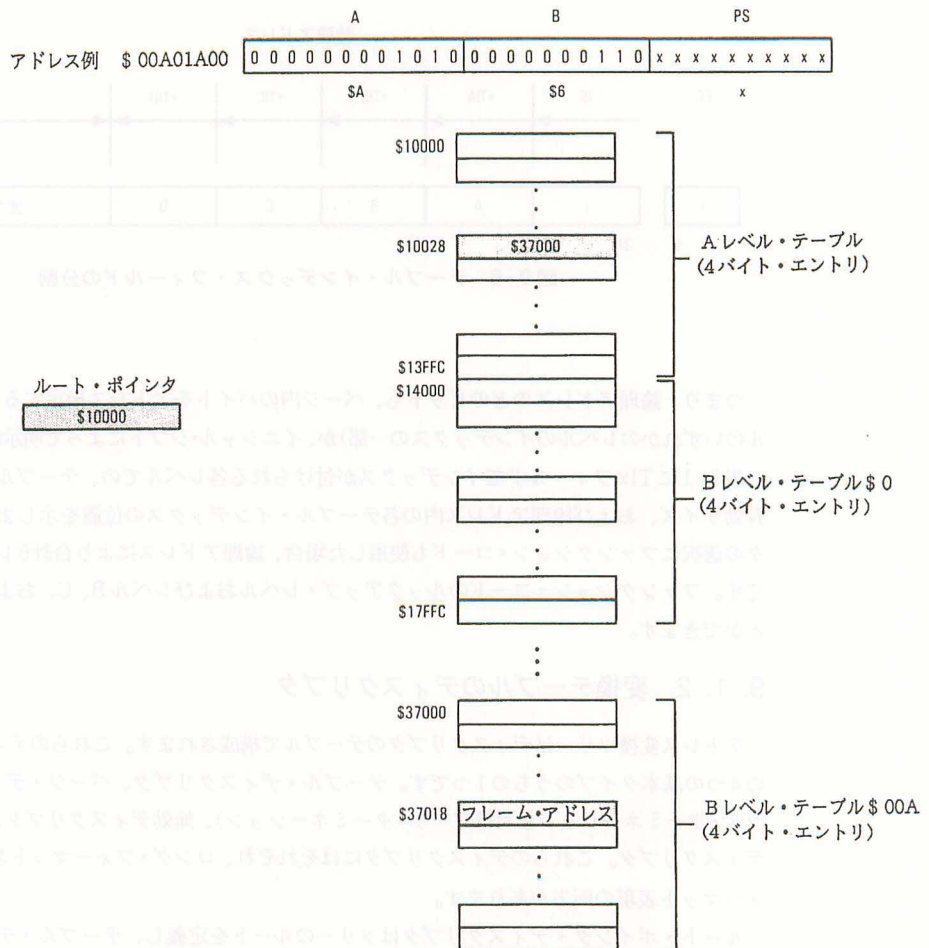



図9-5　メモリ内での変換テーブル・ツリーのレイアウト例

のレベルのポインタ・テーブルには最大32のディスクリプタが含まれます。

同様に、TCレジスタのTIB、TIC、およびTIDフィールドは、下位レベルの変換テーブル・ツリーのインデックスを定義します。これらのフィールドのいずれかがゼロのときには、残りのTIxフィールドは無視されます。最後の非ゼロTIxフィールドは、最下位レベルのツリー構造のインデックスを定義します。このレベルのインデックスで選択されるテーブルはページ・テーブルです。これらのテーブルのすべてのディスクリプタはページ・ディスクリプタです(あるいは、ページ・ディスクリプタを表わします)。図9-6に、TCレジスタのTIxフィールドをファンクション・コードおよび論理アドレスに適用する方法を示します。

たとえば、FCLビットが1にセットされ、TIAフィールドに5、TIBフィールドに9、そしてTICおよびTIDフィールドに0をもつTCレジスタは、3レベルの変換ツリーを定義します。最上位レベルはファンクション・コードでインデックスが付けられ、次のレベルは5つの論理アドレス・ビット、そして最下位レベルは9つの論理アドレス・ビットでそれぞれインデックスが付けられます。TICフィールドに0の代わりに9がある場合、変換ツリーは4レベルをもち、最下位の2レベルはそれぞれ、論理アドレスの9ビット部分でインデックスが付けられます。

TCレジスタのフィールドは、次の等式を満足しなければなりません。

$$IS + PS + TIA + TIB^{1} + TIC^{1} + TID^{1} = 32$$

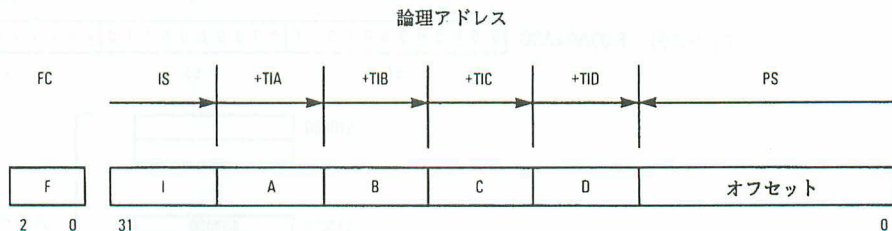



図9-6　テーブル・インデックス・フィールドの分割

つまり、論理アドレスのどのビットも、ページ内のバイトをアドレス指定する(アドレス・テーブルのいずれかのレベルのインデックスの一部)か、イニシャル・シフトによって明示的に無視されます。

表9-1にTIxフィールドでインデックスが付けられる各レベルでの、テーブル・インデックスの有効サイズ、および論理アドレス内の各テーブル・インデックスの位置を示します。ディスクリプタの選択にファンクション・コードも使用した場合、論理アドレスにより合計5レベルまで定義可能です。ファンクション・コードのルックアップ・レベルおよびレベルB、C、およびDは禁止することができます。

9. 1. 2　変換テーブルのディスクリプタ

アドレス変換ツリーはディスクリプタのテーブルで構成されます。これらのディスクリプタは、次の4つの基本タイプのうちの1つです。テーブル・ディスクリプタ、ページ・ディスクリプタ(ノーマル・ターミネーションまたはアーリ・ターミネーション)、無効ディスクリプタ、インダイレクト・ディスクリプタ。これらのディスクリプタにはそれぞれ、ロング・フォーマットおよびショート・フォーマット表現の両方があります。

ルート・ポインタ・ディスクリプタはツリーのルートを定義し、テーブル・ディスクリプタまたはアーリ・ターミネーション・ページ・ディスクリプタを設定することができます。テーブル・ディスクリプタは、変換ツリーで次に低いレベルを定義するメモリ内のディスクリプタ・テーブルを指します。アーリ・ターミネーション・ページ・ディスクリプタはテーブル・サーチを直ちに終了させ、その中には連続論理アドレスに対応するページ・フレームを保持するメモリ内の領域の物理アドレスがあります(「9. 5. 3. 1　アーリ・ターミネーションと連続メモリ」を参照)。

変換ツリーの中間レベルにあるテーブルには、ルート・ポインタ・ディスクリプタに類似したディスクリプタがあります。これらのテーブルには、テーブル・ディスクリプタまたはアーリ・ターミネーション・ページ・ディスクリプタがあり、また無効ディスクリプタをもっていることもあります。

注1　これらのフィールドのいずれかがゼロの場合、それ以降のフィールドは無視されます。

変換ツリーの最下位レベルにあるディスクリプタ・テーブルには、ページ・ディスクリプタ、イ

表9-1　サイズの制限

フィールド	開始ビット位置	サイズの制限
A	31-IS	1-15 (TIAはゼロより大であること。TIB＝0の場合は最小2)
B	31-IS-TIA	0-15
C	31-IS-TIA-TIB	0-15 (TIBがゼロの場合は無視)
D	31-IS-TIA-TIB-TIC	0-15 (TIBまたはTICがゼロの場合は無視)

ンダイレクト・ディスクリプタ、および無効ディスクリプタしかありません。変換ツリーの最下位レベルにあるページ・ディスクリプタは、ページの論理アドレスに対応するメモリ内のページ・フレームの物理アドレスを定義します。インダイレクト・ディスクリプタには実ページ・ディスクリプタを指すポインタがあり、シングル・ページ・ディスクリプタを2つまたはそれ以上の論理アドレスでアクセスするときに使用できます。

変換テーブル設計の柔軟性を高めるため、ディスクリプタ(ルート・ポインタ・ディスクリプタを除く)をショート・フォーマットまたはロング・フォーマットにすることができます。ショート・フォーマット・ディスクリプタは1つのロング・ワードで構成され、インデックスの制限機能やスーパバイザ・オンリ保護はありません。ロング・フォーマット・ディスクリプタは2つのロング・ワードで構成され、MC68030のためのすべての定義済みディスクリプタ・フィールドを含んでいます。ポインタ・テーブルおよびページ・テーブルは、それぞれショート・フォーマットまたはロング・フォーマットのディスクリプタをもつことができますが、1つのテーブル内に両方のサイズを含むことはできません。変換ツリーで異なるレベルにあるテーブルには、さまざまなフォーマットのディスクリプタをもつことができます。同一レベルにあるテーブルにもさまざまなフォーマットのディスクリプタをもつことができますが、特定のポインタ・テーブルまたはページ・テーブルにあるディスクリプタはすべて同一のフォーマットでなければなりません。図9-7にいくつかの異なるフォーマットのディスクリプタを使用する変換ツリーを示します。

ディスクリプタには、すべてにディスクリプタ・タイプ(DT)フィールドがあり、ディスクリプタを識別したり、ディスクリプタが指すテーブルでのディスクリプタのサイズを指定します。これは常に、ディスクリプタの最上位(ショート・フォーマットの場合はその)ロング・ワードの最下位2ビットです。

無効ディスクリプタは、変換ツリーのルート・ポインタ以外の任意レベルで使用できます。通常変換のテーブル・サーチで無効ディスクリプタを検出すると、プロセッサはバス・エラー例外を受け取ります。無効ディスクリプタを使用して、外部デバイスに記憶されているがメモリには存在しないツリーのページや分岐を識別したり、未定義の変換テーブルの一部を識別することができます。これらの2つのケースでは、例外ルーチンがディスクからページを復元したり、変換テーブルに追加することができます。

ロング・フォーマット・ディスクリプタおよびショート・フォーマット無効ディスクリプタにはすべてに、1つまたは2つの未使用フィールドがあります。オペレーティング・システムは、これらのフィールドを独自の目的に使用します。たとえば、オペレーティング・システムはこれらのフィールドをエンコードして無効ディスクリプタのタイプを指定できます。あるいは、主記憶にないページの外部デバイス・アドレスを未使用フィールドに記憶することができます。

9.2　アドレス変換

MMUの機能は、MMUレジスタおよびメモリ内の変換テーブル・ツリーにシステムが記憶する制御情報に基づいて、論理アドレスを物理アドレスに変換することです。

9.2.1 アドレス変換の一般的なフロー

CPU空間アドレス(FC0-FC2＝＄7)は特別なアドレスで、論理アドレスが変換なしに物理アドレスとして直接使用されます。これ以外のアクセスでは、次のように変換処理が進められます。

1. リード・アクセスで必要な命令ワードまたはオペランドに対するオンチップ・データおよび命令キャッシュをサーチする。
2. 論理アドレスおよびファンクション・コードをトランスペアレント変換レジスタのトランスペ

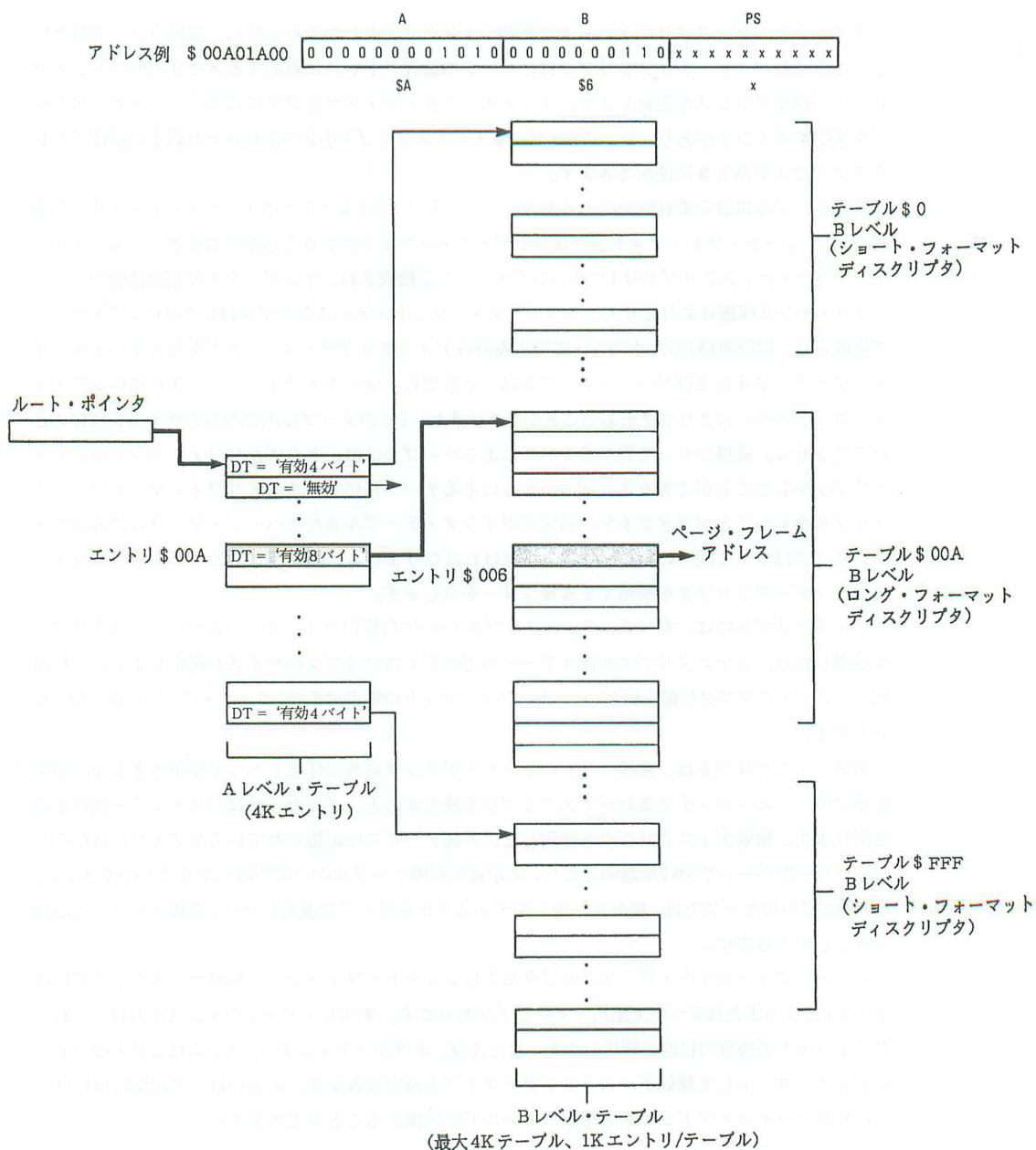



図9-7　各種フォーマットのディスクリプタを使用した変換ツリーの例

アレント変換パラメータと比較し、これらのレジスタのうち一方または両方が一致すれば、この論理アドレスをメモリ・アクセス用の物理アドレスとして使用する。

3. 論理アドレスおよびファンクション・コードをATCのエントリのタグ部分と比較し、一致した場合は、これに対応する物理アドレスをメモリ・アクセスに使用する。
4. (リードで)オンチップ・キャッシュのヒットが起こらず、しかもTTxレジスタの一致がないかまたは有効なATCエントリの一致がない場合は、テーブル・サーチ動作を開始して対応する変換ツリーから物理アドレスを獲得し、論理アドレスに対する有効なATCエントリを生成し、ステップ3を繰り返す。

図9-8 はアドレス変換の一般的なフローチャートです。フローチャートの最上段の分岐は、CPU空間(FC0-FC2＝＄7)へのアクセスに適用されます。その次の分岐はリード・アクセスにのみ適用されます。オンチップ・キャッシュのうちどちらかでもヒットすれば(必要なデータまたは命令がある)、メモリ・アクセスは不要です。3番目の分岐はトランスペアレント変換に適用されます。最後の3つの分岐は、次のようにATC変換に適用されます。要求されたアクセスがATC内でミスすれば、メモリ・サイクルがアボートされ、テーブル・サーチ動作が継続されます。テーブル・サーチの後でATCエントリが生成され、アクセスが再試行されます。ATC内でアクセスがヒットしても、そのATCエントリを生成したテーブル・サーチ中にバス・エラーが発見された場合、そのメモリ・アクセスはアボートされ、バス・エラー例外が発生します。

アクセスの結果ATCがヒットしても、アクセスがライト・アクセスまたはリード・モディファイ・ライト・アクセスで、このページが書込み保護されている場合、このメモリ・サイクルもやはりアボートされ、バス・エラー例外が発生します。ライト・アクセスまたはリード・モディファイ・ライト・アクセスの場合は、ATCエントリの修正ビットがセットされていなければ、メモリ・サイクルはアボートされ、メモリのページ・ディスクリプタおよびATCの両方にある修正ビットをセットするようテーブル・サーチが行なわれ、アクセスが再試行されます。ATCエントリの修正ビットがセットされており、バス・エラー・ビットがセットされてなく、TTxレジスタが一致せず、CPU空間へのアクセスではないと仮定すれば、ATCは次の2つの条件のもとでバス・コントローラに対してアドレス変換を行ないます。1)どちらのオンチップ・キャッシュでもリード・アクセスがヒットしない場合、 2)ライト・アクセスまたはリード・モディファイ・ライト・アクセスが書込み保護されていない場合。

前述した一般的なフローチャートについての説明では、メモリ・サイクルをアボートさせる条件をいくつか指定しています。このような場合、ASがアサートされる前にバス・サイクルがアボートされます。

9.2.2 $\overline{\mathrm{RESET}}$がMMUに及ぼす影響

$\overline{\mathrm{RESET}}$信号をアサートしてMC68030をリセットすると、TCレジスタおよびTTxレジスタのEビットがクリアされてアドレス変換をディセーブルします。これにより、論理アドレスが物理アドレスとして、バス・コントローラに渡されます。そのため、オペレーティング・システムは必要に応じて変換テーブルとMMUレジスタをセットアップすることができます。変換テーブルとレジスタの初期設定が終了した後、TCレジスタのEビットをセットすれば、アドレス変換をイネーブルにすることができます。プロセッサをリセットしてもATCのエントリは無効化されません。リセット操作の後、変換をイネーブルする前に、ATCから存在するすべての有効エントリをフラッシュするために、ATCをフラッシュするMMU命令(たとえばPMOVE)を実行しなければなりません。

9.2.3 $\overline{\mathrm{MMUDIS}}$がアドレス変換に及ぼす影響

$\overline{\mathrm{MMUDIS}}$をアサートすると、MMUはATCのサーチを、そして実行ユニットはテーブル・サーチを実行できなくなります。アドレス変換がディセーブルされているときには、論理アドレスが物理アドレスとして使用されます。アドレス指定されたポート・サイズより大きく、ロング・ワード・アラインメントされたデータ・オペランドに最初のアクセスを行なうと、オペランドのそれ以外の部分に対する以降のバス・サイクルでは、最初のバス・サイクルで使用したのと常に同じ上位アドレス・ビット(適宜A0およびA1を変更)を使用します。したがって、この種の操作中に$\overline{\mathrm{MMUDIS}}$がアサートされた場合、すべての転送が完了するまでは、アドレス変換のディセーブルは有効とはなりません。なお、$\overline{\mathrm{MMUDIS}}$をアサートしてもトランスペアレント変換レジスタの動作に影響を与えることはありません。

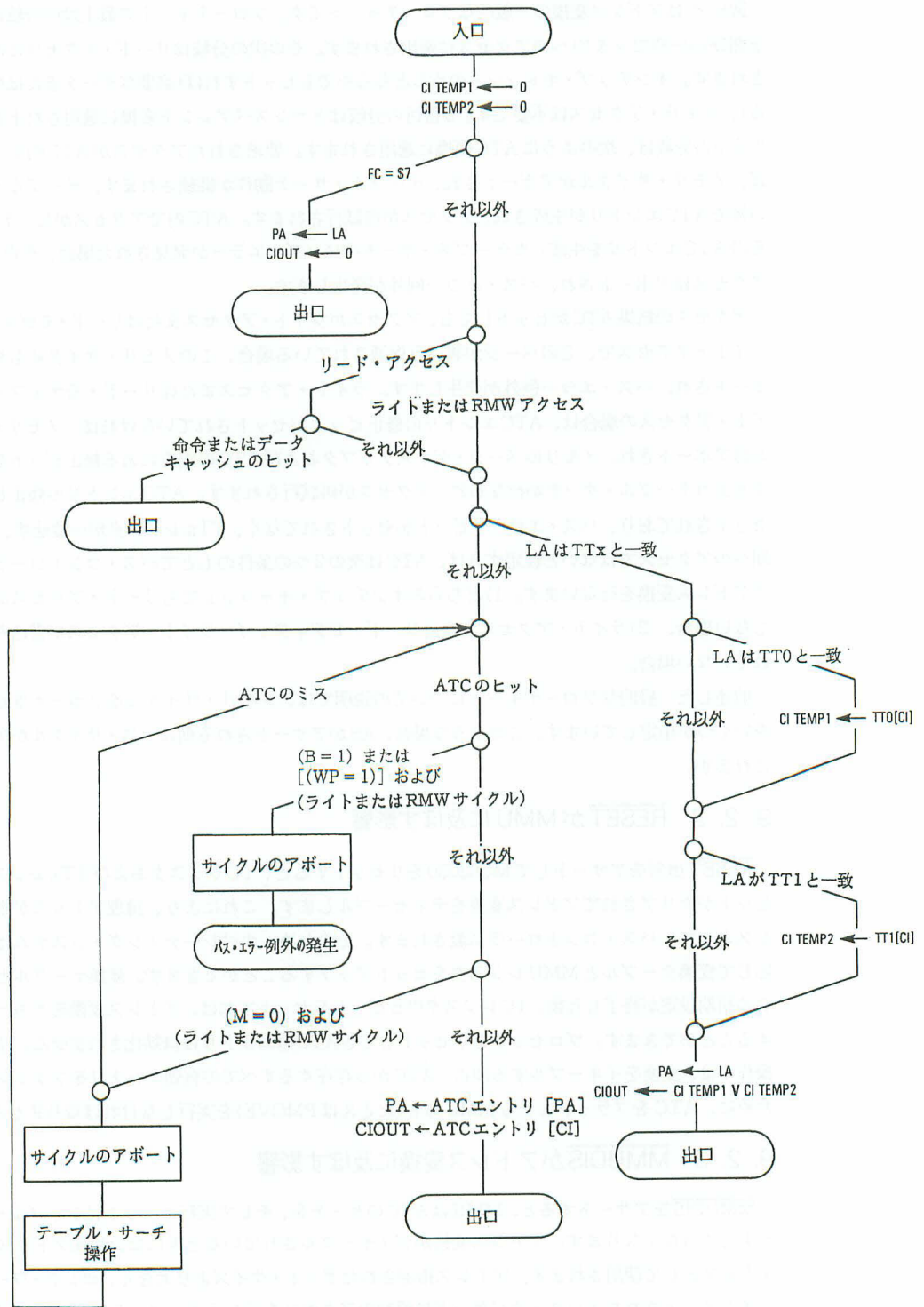



図9-8 アドレス変換の一般的なフローチャート

9.3　トランスペアレントな変換

MMUにある2つの独立したトランスペアレント変換レジスタ(TT0およびTT1)は、オプションにより直接物理アドレス空間に変換する論理アドレス空間を、2ブロック定義することができます。MMUはこれらのブロックのアドレスに対する書込み保護を、明示的にチェックすることはありませんが、ブロックをリード・サイクルについてのみトランスペアレントであると指定することができます。TTxレジスタが定義したアドレスのブロックには、最低16Mバイトの論理アドレス空間があります。2つのブロックはオーバラップしていてもよく、また分離することもできます。

以下に述べるアドレス比較に関する説明では、TT0とTT1は両方ともイネーブルされているものと想定しています。しかし、それぞれのTTxレジスタは個別にディセーブルすることができます。ディセーブルされたTTxレジスタは完全に無視されます。

MMUが変換すべきアドレスを受け取ると、ファンクション・コードとアドレスの上位8ビットを、TT0およびTT1で定義したアドレスのブロックと比較します。各TTxレジスタのアドレス空間ブロックは、ベース・ファンクション・コード、ファンクション・コード・マスク、論理ベース・アドレス、および論理アドレス・マスクで定義されます。マスク・フィールドのビットがセットされていると、ベース・ファンクション・コードまたは論理ベース・アドレスの対応するビットは、ファンクション・コードおよびアドレス比較では無視されます。アドレス・マスクで上位ビットを連続してセットすれば、トランスペアレントに変換されるブロックのサイズが増加します。

ファンクション・コード・ビットとアドレス・ビット(マスクされたビットを除く)が等しい場合は、現在のバス・サイクルのアドレスとTTxレジスタのアドレスが一致したことになります。各TTxレジスタはリード・アクセスやライト・アクセスをトランスペアレントとして指定できます。その場合、一致が発生するには、内部のリード/ライト信号がTTxレジスタのR/$\overline{\mathrm{W}}$ビットと一致していなければなりません。アクセスのタイプ(リードまたはライト)の選択もマスクすることができます。リード・モディファイ・ライト操作を行なう命令で使用するアドレスをトランスペアレントに変換するには、リード/ライト・マスク・ビット、RWMをセットしておかなければなりません。そうしないと、リード・モディファイ・ライト操作での個々のサイクルに対するファンクション・コードおよびアドレス・ビットには関係なく、リード・モディファイ・ライト操作のリード部分およびライト部分のどちらも、TTxレジスタでトランスペアレントにマップされません。

トランスペアレント変換レジスタを適当にコンフィギュレーションすることで、柔軟なトランスペアレント・マッピングを指定できます。たとえば、TTxレジスタを用いてユーザ・プログラム空間をトランスペアレントに変換するには、このレジスタのRWMビットを1に、FC ベースを＄2に、そしてFC MASKを＄0にセットします。アドレス＄00000000－＄0FFFFFFFのスーパバイザ・データ・リード・アクセスをトランスペアレントに変換するには、論理ベース・アドレス・フィールドを＄0Xにセットし、論理アドレス・マスクを＄0F、R/$\overline{\mathrm{W}}$ビットを1、RWMビットを0、FCベースを＄5、FCマスク・フィールドを＄0にそれぞれセットします。この例ではTTxレジスタはリード・サイクルしか指定していませんので、このアドレス範囲へのライト・アクセスを変換テーブルで変換し、必要に応じて書込み保護を設定することができます。

各TTxレジスタは、そのブロックの論理アドレスの内容を内部キャッシュまたは外部キャッシュのどちらにも記憶させないように指定できます。あるアドレスがTTxレジスタで指定するアドレスと一致し、そのTTxレジスタのキャッシュ・インヒビット・ビットがセットされているときには、キャッシュ・インヒビット出力信号($\overline{\mathrm{CIOUT}}$)がアサートされます。オンチップ命令およびデータ・キャッシュが$\overline{\mathrm{CIOUT}}$を使用して、このアドレスに関連するデータのキャッシングを禁止します。外部キャッシュも同じ目的でこの信号を利用できます。

アクセスに関しては、これらのレジスタのどちらかが一致すれば、アクセスはトランスペアレン

トに変換されます。両方のレジスタが一致すれば、CIビットのORをとって$\overline{\text{CIOUT}}$信号を生成します。

ページの物理アドレスが論理アドレスと同じに設定されている場合、変換ツリーの変換テーブルでもトランスペアレント変換を実現できます。

9.4 アドレス変換キャッシュ

アドレス変換キャッシュ(ATC)は、22エントリのフル・アソシエイティブ(連想記憶型)キャッシュです。最も近い時点で使用された論理アドレスを高速にアドレス変換するために、メモリ内の対応するページ・ディスクリプタと同様な形式のアドレス変換情報を用意しています。

MC68030は、ATCの変換時間が他の操作によって常に完全にオーバラップされ、ATCのサーチに性能上のペナルティがないように編成されています。アドレス変換は外部バス・サイクルを開始する前に、オンチップ命令およびデータ・キャッシュへのアクセスと並行して行なわれます。

可能であれば、ATCは新しいアドレス変換値を記憶したときに、すでに有効ではなくなったエントリを交換します。ATC内のすべてのエントリが有効である場合、ATCは擬似的なleast-recently-used(最も古い時点での使用)アルゴリズムを用いて、交換すべき有効エントリを選択します。ATCは有効性ビットおよび内部ヒストリ・ビットを使用してこの交換アルゴリズムを実現します。ATCのヒット率はアプリケーションによって異なりますが、98%から99%以上が期待できます。

各ATCエントリには、論理アドレス、および物理アドレスをもつ対応するページ・ディスクリプタからの情報があります。各エントリの28ビットの論理(またはタグ)部分は、次の3つのフィールドからなります。

27	26　　24	23　　　　　　　　　　　　0
V	FC	論理アドレス

V——有効

このビットはエントリの妥当性を示します。Vがセットされていれば、このエントリは有効です。このビットはMC68030がエントリをロードするときにセットされます。フラッシュ操作によりクリアされます。次の操作のいずれもエントリのVビットをクリアします。

- CRP、SRP、TC、TT0またはTT1レジスタに値をロードするFDビットがゼロのPMOVE命令
- PFLUSHA命令
- このエントリを選択するPFLUSH命令
- 論理アドレスに対するPLOAD命令、およびこのエントリ用のタグに一致するFC。この命令は、指定された論理アドレスに対して、(Vビットをセットして)新しいエントリを書き込みます。
- ATCの置換アルゴリズムによる置換を目的としたエントリの選択。

FC——ファンクション・コード

この3ビット・フィールドには、このエントリの論理アドレスに対応するファンクション・コード・ビット(FC0-FC2)があります。

論理アドレス

この24ビット・フィールドには、このエントリに対する最上位論理アドレス・ビットがあります。ページ・サイズが256バイトのとき、このフィールドの24ビットすべてを使用して、このエントリと入力論理アドレスを比較します。ページ・サイズがこれより大きい

場合、このフィールドの最下位ビットが適当な数だけ無視されます。

エントリの各論理部分には、対応する28ビットの物理(またはデータ)部分があります。物理部分には次のフィールドがあります。

27	26	25	24	23　　　　　　　　　　　　　　　　　　0
B	CI	WP	M	物理アドレス

B——バス・エラー

エントリに対応するテーブル・サーチ中に、バス・エラー、無効ディスクリプタ、スーパバイザ違反、またはリミット違反が見つかると、そのエントリに対してこのビットがセットされます。Bがセットされたとき、それ以降この論理アドレスにアクセスするとMC68030はバス・エラー例外を受け取ります。ATCミスはテーブル・サーチ操作後、直ちにアクセスの再試行を引き起こすため、バス・エラー例外は再試行時に発生します。このエントリに対するPFLUSH命令またはPLOAD命令がこのエントリを無効にするか、またはATCに対する交換アルゴリズムによって交換されるまでは、Bビットはセットされたままです。

CI——キャッシュ・インヒビット

このビットは、このエントリに対応するページ・ディスクリプタのキャッシュ・インヒビット・ビットがセットされるとセットされます。MC68030がCIビットがセットされているエントリの論理アドレスにアクセスしたときに、対応するバス・サイクルでキャッシュ・インヒビット出力信号($\overline{\text{CIOUT}}$)をアサートします。この信号はオンチップ・キャッシュでのキャッシングを禁止するもので、外部キャッシュに使用することもできます。

WP——書込み保護

このビットは書込み保護ビットです。このエントリのテーブル・サーチ中に見つかったディスクリプタのWPビットがセットされているときにセットされます。テーブル・ディスクリプタのWPビットをセットすれば、そのディスクリプタでアクセスされるすべてのページに書込み保護がなされます。WPビットがセットされているときは、このエントリに対応する論理アドレスへの、ライト・アクセスまたはリード・モディファイ・ライト・アクセスによって、直ちにバス・エラー例外が発生します。

M——修正済み

このビットは修正済みビットです。エントリに対応する論理アドレスへの有効なライト・アクセスが発生したときにセットされます。Mビットがクリアされているときに、この論理アドレスへライト・アクセスが行なわれると、MC68030はこのアクセスをアボートし、ページ・ディスクリプタのMビットをセットして、古いATCエントリを無効化し、Mビットをセットした新しいエントリを生成して、テーブル・サーチを開始します。MMUは次に、最初のライト・アクセスの実行を許可します。これによって、あるページに対する以前のリード操作でMビットをクリアして、ATC内にそのページに対するエントリが生成された場合であっても、該当ページに対する最初のライト操作によってATCおよび変換テーブル内のページ・ディスクリプタの両方でMビットが確実にセットされることになります。

物理アドレス

この24ビット・フィールドには、論理アドレスに対応するページ・ディスクリプタからの物理アドレス・ビット(A8-A31)があります。ページ・サイズが256バイトを超えるときには、物理アドレス・ビットの全ビットが使用されるわけではありません。論理アドレスのすべてのページ・インデックス・ビットは、変換しないでバス・コントローラに転送さ

れます。

9. 5　変換テーブルの詳細

変換テーブルの詳細およびそれらの使用法には、ディスクリプタ、テーブル・サーチ、変換テーブル構造のバリエーション、およびMC68030 MMUで利用できる保護手法の詳細な説明が含まれています。

9. 5. 1　ディスクリプタの詳細

ディスクリプタの詳細説明には、変換ツリーで使用するショート・フォーマットおよびロング・フォーマットのディスクリプタに関する詳細な説明があります。すべてのディスクリプタに適用できるフィールドを最初の項で説明します。

9. 5. 1. 1 ディスクリプタ・フィールドの定義

すべてのディスクリプタ・フィールドは、複数のタイプのディスクリプタで使用されます。この項ではこれらのフィールドを列記し、各フィールドの使用法を説明します。

DT

この2ビット・フィールドにはディスクリプタ・タイプが含まれています。最初の2つのタイプはディスクリプタ自体に適用されます。他の2つのタイプはツリーの次のレベルにあるテーブルのディスクリプタに適用されます。この値の定義は次のとおりです。

＄0　無効

このコードは現在のディスクリプタを無効ディスクリプタとして識別します。無効なディスクリプタが見つかると、テーブル・サーチが終了します。

＄1　ページ・ディスクリプタ

このコードは現在のディスクリプタをページ・ディスクリプタとして識別します。ページ・ディスクリプタがページ・テーブル(変換ツリーの最下位レベルにある)にある場合は、正常なページ・ディスクリプタです。これより上位にあるページ・ディスクリプタは、アーリ・ターミネーション・ページ・ディスクリプタとして定義されます。いずれかのタイプのページ・ディスクリプタが見つかると、テーブル・サーチが終了します。

＄2　有効4バイト

このコードは、次にアクセスされるテーブルにショート・フォーマット・ディスクリプタがあることを指定します。MC68030は次のテーブルのインデックスを4倍して次のディスクリプタにアクセスします(ショート・フォーマット・ディスクリプタはロング・ワードに整列していなければならない)。ページ・テーブル(変換ツリーの最下位レベル)で使用した場合、このコードはショート・フォーマット・ページ・ディスクリプタを指す間接ディスクリプタを識別します。

＄3　有効8バイト

このコードは、次にアクセスされるテーブルにロング・フォーマット・ディスクリプタがあることを指定します。MC68030は次のテーブルのインデックスを8倍して次のディスクリプタにアクセスします(ロング・フォーマット・ディスクリプタはクワッド・ワードに整列していなければならない)。ページ・テーブル(変換ツリーの最下位レベル)で使用した場合、このコードはロング・フォーマット・ページ・ディスクリプタを指す間接ディスクリプタを識別します。

U

このビットは、スーパバイザ違反が検出された後を除いて、ディスクリプタがアクセスされたときにUビットがクリア状態のとき、プロセッサによって自動的にセットされます。ページ・ディ

スクリプタ・テーブルでは、このビットをセットして、ディスクリプタに対応するページがアクセスされたことを示します。ポインタ・テーブルでは、このビットをセットして、MC68030がテーブル・サーチの一部としてこのポインタにアクセスしたことを示します。なお、ツリーの別のレベルでアクセスが拒否されるようなアドレスに対しては、ポインタがフェッチされ、そのUビットがセットされることがあります。MC68030がページのアクセスを許可する前に、Uビットが更新されます。プロセッサがこのビットをクリアすることはありません。

WP

このビットにより書込み保護が行なわれます。テーブル・サーチ中に出会ったすべてのWPビットの状態は論理的にORされ、その結果が論理アドレスに対するテーブル・サーチの終わりで、ATCエントリにコピーされます。PTEST命令のテーブル・サーチ中には、プロセッサはこの結果をMMUステータス・レジスタ(MMUSR)にコピーします。

WPがセットされると、MC68030は他のプログラムがそのディスクリプタでマップされる論理アドレス空間に書込みを行なうことを禁止します(つまり、この保護は絶対的なものです)。WPビットがクリアされている場合、MC68030はこのディスクリプタを使用して(アクセスが変換ツリーのあるレベルで制限されている場合を除いて)ライト・アクセスを許可します。

CI

このビットがセットされると、オンチップの命令キャッシュおよびデータ・キャッシュによるこのページ内のアイテムのキャッシングを禁止し、MC68030にこのページ内のアイテムをアクセスするバス・サイクルに対して$\overline{\text{CIOUT}}$信号をアサートさせます。

L/U

このビットはリミット・フィールドのリミットのタイプを指定します。L/Uビットがセットされると、リミット・フィールドには符号なしの下限値が入ります。次のレベルのツリーに対するインデックス値は、リミット・フィールド値以上でなければなりません。このビットがクリアされると、リミットは符号なしの上限値となり、インデックス値はリミット以下でなければなりません。境界外のアクセスを行なうと、そのアドレスに対するATCエントリのBビットがセットされ、テーブル・サーチはアボートされます。

リミット

この15ビット・フィールドには、アドレスのインデックス部分と比較して境界外のインデックスを検出するためのリミット値が含まれています。リミット・チェックは、変換ツリーで次のレベルにあるテーブルへのインデックスに適用されます。ディスクリプタがアーリ・ターミネーション・ページ・ディスクリプタの場合、このリミット・フィールドは論理アドレスの次のインデックス・フィールドに対するチェックとして使用されます。

M

このビットは修正済みのページを識別します。WPビットがセットされたディスクリプタに出会うか、スーパバイザ違反を検出した後を除いて、MC68030はMビットがゼロになっているページへの書込み操作を行なう前に、対応するページ・ディスクリプタのMビットをセットします。R/$\overline{\text{W}}$または$\overline{\text{RMC}}$信号が"L"の場合、アクセスは更新を目的とする書込みであるとみなされます。MC68030がこのビットをクリアすることはありません。

ページ・アドレス

この24ビット・フィールドには、メモリ内のページの物理ベース・アドレスが含まれています。このアドレスの下位ビットは論理アドレスで与えられます。ページ・サイズが256バイトを超える場合、このフィールドの最下位ビットのうち1ビット以上が使用されません。未使用ビット数は、TCレジスタのPSフィールドの値から8を引いた値に等しくなります。

S

このビットはポインタ・テーブルまたはページを、スーパバイザ専用テーブルまたはページとして識別します。Sビットがセットされると、スーパバイザの特権レベルで動作しているプログラムだけが、このディスクリプタでマップされる論理アドレス部分へのアクセスを許可されます。このビットがクリアされていると、変換ツリーの他のレベルによってアクセスが制限されていないかぎり、このディスクリプタを用いたアクセスはスーパバイザ専用に制限されることはありません。

テーブル・アドレス

この28ビット・フィールドには、ディスクリプタ・テーブルの物理ベース・アドレスが含まれています。アドレスの下位ビットは論理アドレスによって与えられます。

ディスクリプタ・アドレス

この30ビット・フィールドには、ページ・ディスクリプタの物理アドレスが含まれています。このフィールドはショートおよびロング・フォーマット・インダイレクト・ディスクリプタでのみ使用されます。

未使用

このフィールドのビットは、MC68030は使用せずシステム・ソフトウェアが独自の目的で使用できます。

予約

"1" または"0" で示すディスクリプタ・フィールドは、将来使用するためにモトローラによって予約されています。これらのビットは目的に応じ、一貫して "1" または "0" に書き込んでおかなければなりません。ルート・ポインタでは、これらのビットは変更できません。メモリに常駐しているディスクリプタでは、MC68030はこれらのフィールド値をチェックせず変更もしません。どのような目的であろうと、これらのビットをシステム・ソフトウェアで使用すると、将来の製品でサポートされないことがあります。

9. 5. 1. 2 ルート・ポインタ・ディスクリプタ

ルート・ポインタ・ディスクリプタには、変換ツリーの最上位レベルのポインタ・テーブルのアドレスが含まれています。このタイプのディスクリプタは、PMOVE命令によってCRPおよびSRPレジスタにロードされます。前項のフィールドの説明は、2つのわずかな例外を除いて、CRPおよびSRPの対応するフィールドにも適用されます。$ 00(無効)のディスクリプタ・タイプ・コードは許されません。CRPまたはSRPレジスタのDTフィールドにゼロをロードしようとすると、MMUコンフィギュレーション例外になります。また、TCレジスタのFCLフィールドがセットされていると、ルート・ポインタ・レジスタのL/Uおよびリミット・フィールドは使用されません。図9-9にルート・ポインタ・ディスクリプタのフォーマットを示します。

9. 5. 1. 3 ショート・フォーマット・テーブル・ディスクリプタ

「9. 5. 1. 1 ディスクリプタ・フィールドの定義」で述べたフィールドの説明が、このディスクリプタの対応するフィールドにも適用されます。図9-10にショート・フォーマット・テーブル・ディスクリプタのフォーマットを示します。

9. 5. 1. 4 ロング・フォーマット・テーブル・ディスクリプタ

「9. 5. 1. 1 ディスクリプタ・フィールドの定義」で述べたフィールドの説明が、このディスクリプタの対応するフィールドにも適用されます。アドレスの計算中、MC68030は未使用フィールドを内部でゼロに置き換えます。図9-11 にロング・フォーマット・テーブル・ディスクリプタのフォーマットを示します。

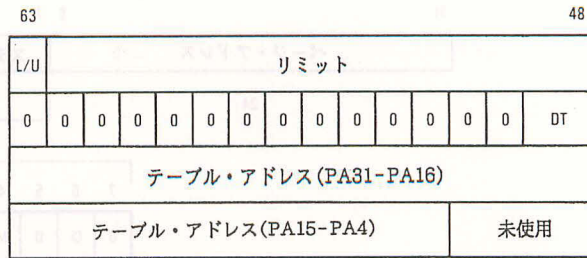



図9-9　ルート・ポインタ・ディスクリプタのフォーマット

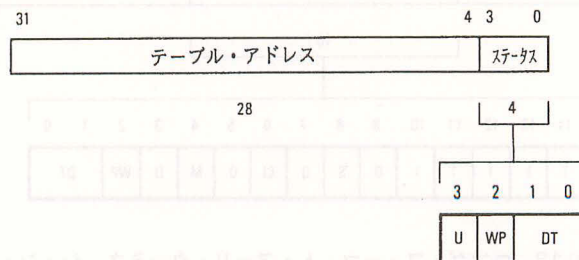



図9-10　ショート・フォーマットのテーブル・ディスクリプタ

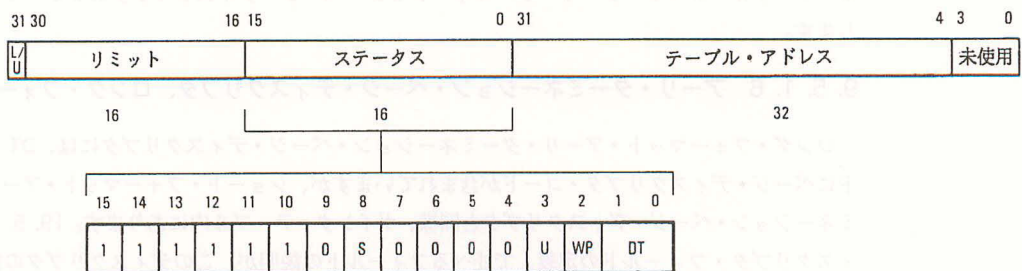



図9-11　ロング・フォーマットのテーブル・ディスクリプタ

9.5.1.5 アーリ・ターミネーション・ページ・ディスクリプタ、ショート・フォーマット

ショート・フォーマット・アーリ・ターミネーション・ページ・ディスクリプタには、DTフィールドにページ・ディスクリプタ・コードが含まれていますが、これはポインタ・テーブル内にあります。

つまり、アーリ・ターミネーション・ページ・ディスクリプタがあるテーブルは、アドレス変換ツリーの最下位レベルにはありません。「9.5.1.1 ディスクリプタ・フィールドの定義」で述べるフィールドの説明が、このディスクリプタの対応するフィールドにも適用されます。図9-12にシ

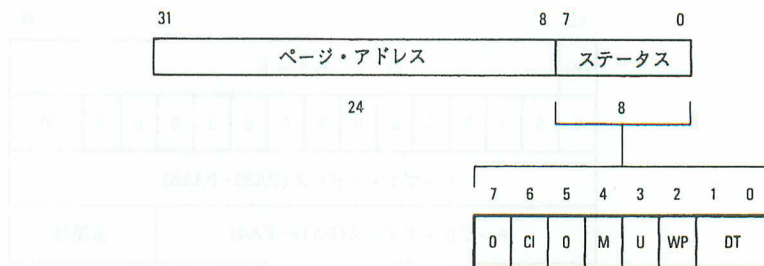



図9-12　ショート・フォーマットのページ・ディスクリプタおよびショート・フォーマットのアーリ・ターミネーション・ページ・ディスクリプタ

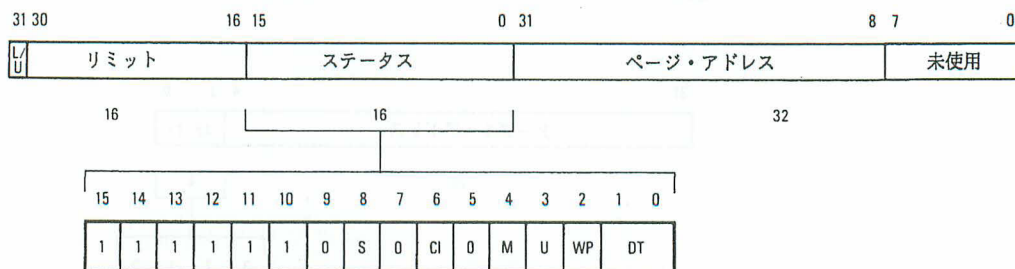



図9-13　ロング・フォーマット・アーリ・ターミネーション・ページ・ディスクリプタ

ョート・フォーマット・アーリ・ターミネーション・ページ・ディスクリプタのフォーマットを示します。

9. 5. 1. 6　アーリ・ターミネーション・ページ・ディスクリプタ、ロング・フォーマット

ロング・フォーマット・アーリ・ターミネーション・ページ・ディスクリプタには、DTフィールドにページ・ディスクリプタ・コードが含まれていますが、ショート・フォーマット・アーリ・ターミネーション・ページ・ディスクリプタと同様、ポインタ・テーブル内にあります。「9. 5. 1. 1　ディスクリプタ・フィールドの定義」で述べるフィールドの説明が、このディスクリプタの対応するフィールドにも適用されます。

図9-13にロング・フォーマット・アーリ・ターミネーション・ページ・ディスクリプタのフォーマットを示します。ロング・フォーマット・ディスクリプタのリミット・フィールドにより、ディスクリプタが適用されるページ数を制限することができます。

9. 5. 1. 7　ページ・ディスクリプタ、ショート・フォーマット

ショート・フォーマット・ページ・ディスクリプタはページ・テーブル(アドレス・テーブルの最下位レベル)で使用されます。「9. 5. 1. 1　ディスクリプタ・フィールドの定義」で述べたフィールドの説明が、このディスクリプタの対応するフィールドにも適用されます。ショート・フォーマット・ページ・ディスクリプタは、図9-12に示すショート・フォーマットのアーリ・ターミネーション・ページ・ディスクリプタのものと同じです。

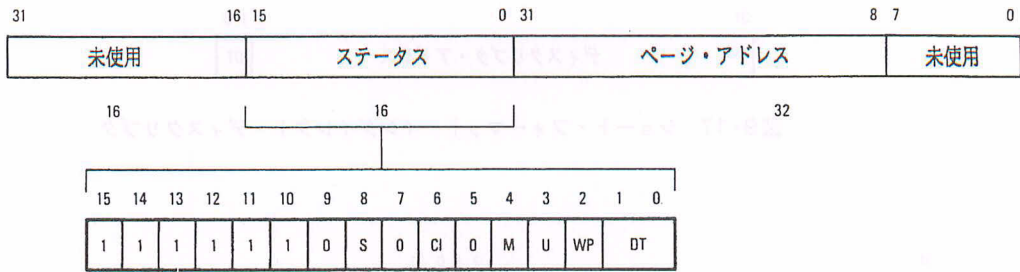



図9-14　ロング・フォーマット・ページ・ディスクリプタ

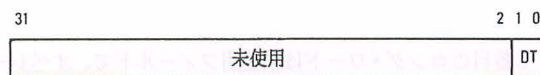



図9-15　ショート・フォーマット無効ディスクリプタ

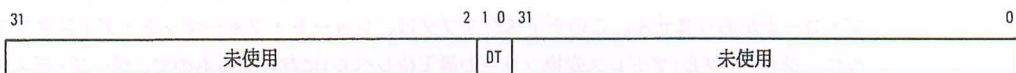



図9-16　ロング・フォーマット無効ディスクリプタ

9. 5. 1. 8　ページ・ディスクリプタ、ロング・フォーマット

ロング・フォーマット・ページ・ディスクリプタもページ・テーブルで使用されます。「9. 5. 1. 1　ディスクリプタ・フィールドの定義」で述べたフィールドの説明が、このディスクリプタの対応するフィールドにも適用されます。図9-14にロング・フォーマット・ページ・ディスクリプタのフォーマットを示します。

9. 5. 1. 9　無効ディスクリプタ、ショート・フォーマット

ショート・フォーマット無効ディスクリプタは、ゼロのDTフィールドだけで構成され、無効ディスクリプタとして識別されます。これは、ルート・ポインタ・レベルを除くどのレベルのアドレス変換ツリーでも使用できます。オペレーティング・システムは、未割当てのテーブル部分または外部デバイス上のテーブル部分を識別するために、30ビットの未使用フィールドを使用できます。たとえば、ディスクに常駐するテーブルまたはページのディスク・アドレスをこのフィールドに格納しておくことができます。図9-15にショート・フォーマット無効ディスクリプタのフォーマットを示します。

9. 5. 1. 10　無効ディスクリプタ、ロング・フォーマット

ロング・フォーマット無効ディスクリプタは、ロング・フォーマット・ディスクリプタをもつポインタおよびページ・テーブルで使用されます。これは、前項のショート・フォーマット無効ディスクリプタと同様に使用されます。最初のロング・ワードには、最下位ビット群にDTフィールドが

図9-17　ショート・フォーマット・インダイレクト・ディスクリプタ

31	2 1 0	31	2 1 0
未使用	DT	ディスクリプタ・アドレス	UN

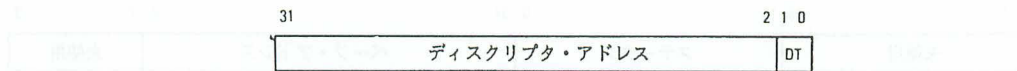
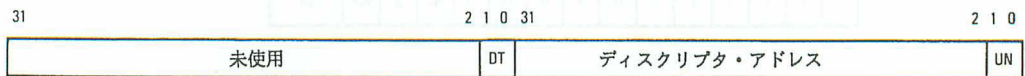



図9-18　ロング・フォーマット・インダイレクト・ディスクリプタ

あります。2番目のロング・ワードは未使用フィールドで、オペレーティング・システムが使用できます。図9-16にロング・フォーマット無効ディスクリプタのフォーマットを示します。

9. 5. 1. 11　インダイレクト・ディスクリプタ、ショート・フォーマット

ショート・フォーマット・インダイレクト・ディスクリプタには、固有のディスクリプタ・タイプ・コードがありません。このディスクリプタは、ショート・フォーマット・ディスクリプタをもつページ・テーブル(アドレス変換ツリーの最下位レベル)に存在するもので、ページ・ディスクリプタでも無効ディスクリプタでもありません。ディスクリプタ・タイプ・フィールドには、参照されるページ・ディスクリプタのサイズにより、有効4バイト・ディスクリプタまたは有効8バイト・ディスクリプタのコードがあります。「9. 5. 1. 1　ディスクリプタ・フィールドの定義」で述べたフィールドの説明が、このディスクリプタの対応するフィールドにも適用されます。図9-17にショート・フォーマット・インダイレクト・ディスクリプタのフォーマットを示します。

9. 5. 1. 12　インダイレクト・ディスクリプタ、ロング・フォーマット

ロング・フォーマットのインダイレクト・ディスクリプタには、前項で説明したショート・フォーマット・インダイレクト・ディスクリプタのすべての属性があります。唯一の違いは、これがロング・フォーマット・ディスクリプタをもつページ・テーブルで使用されるということと、2つの未使用フィールドがあるということです。「9. 5. 1. 1　ディスクリプタ・フィールドの定義」で述べるフィールドの説明が、このディスクリプタの対応するフィールドにも適用されます。図9-18にロング・フォーマット・インダイレクト・ディスクリプタを示します。

9. 5. 2　一般的なテーブル・サーチ

アドレス変換キャッシュ(ATC)に、プロセッサ・アクセスの論理アドレスのためのディスクリプタがなく、しかも変換が必要なときは、MC68030はメモリ内の変換テーブルをサーチし、その論理アドレスに対応するページの物理アドレスとステータス情報を取得します。テーブル・サーチが必要なとき、CPUは命令の実行活動を中断し、テーブル・サーチが正常に終了すると、アドレス・マッピングをATCに格納してアクセスを再試行します。その後アクセスがマッチ(ヒット)し、(例外が見つからなかった場合)変換されたアドレスがバス・コントローラに転送されます。

テーブル・サーチは、表9-2に示すように、TCレジスタのファンクション・コード・ビットFC2およびSREビットを使用して変換ツリーを選択することにより開始されます。SREビットがセットされると、スーパバイザのルート・ポインタがイネーブルされ、ファンクション・コード・ビット

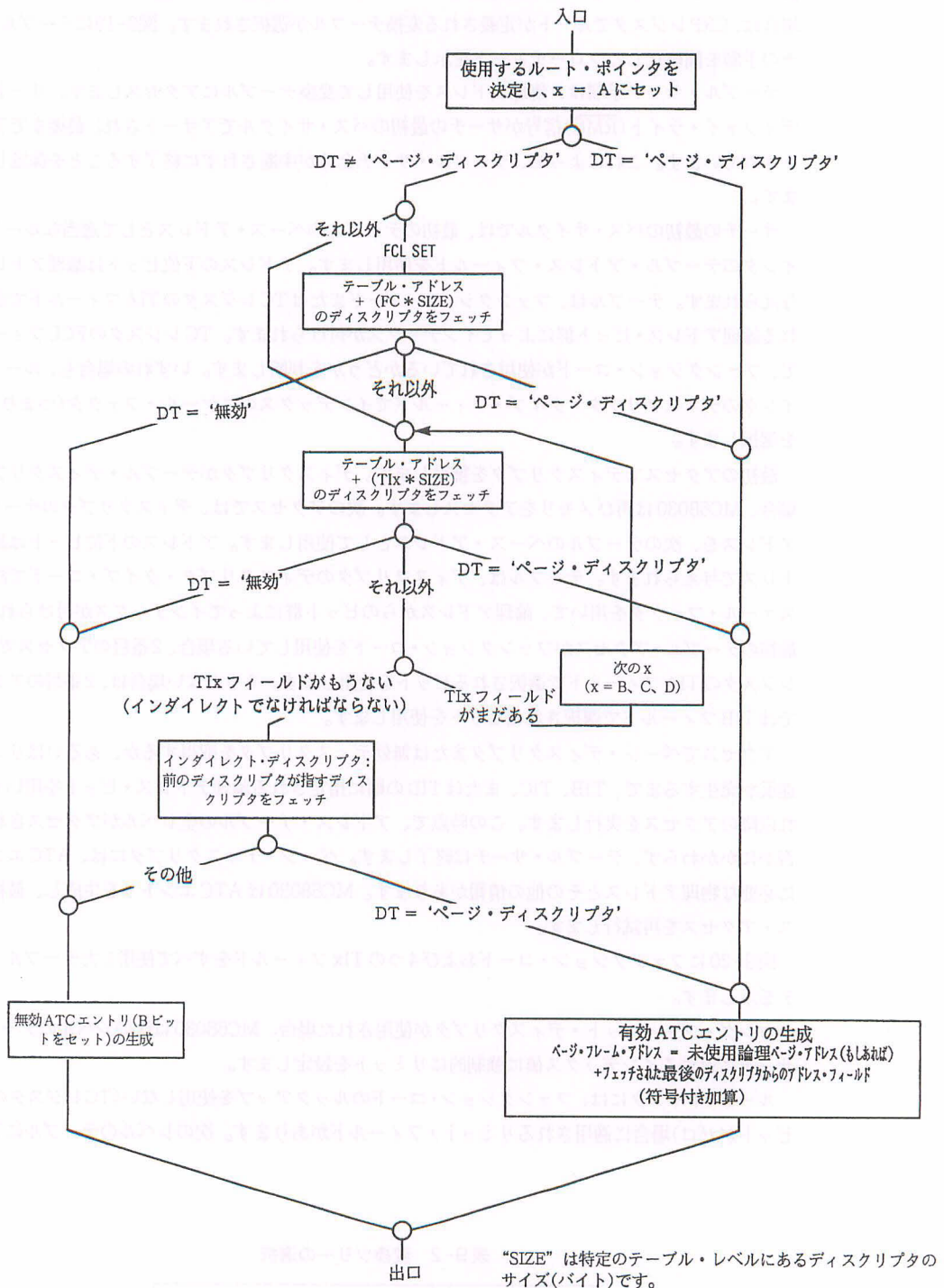



図9-19　単純化したテーブル・サーチのフローチャート

FC2がスーパバイザ・レベルのアクセスにセットされます。ルートがSRPレジスタで定義される変換ツリーは、SREおよびFC2が両方ともにセットされている場合にのみ選択されます。それ以外の場合は、CRPレジスタでルートが定義される変換テーブルが選択されます。図9-19にテーブル・サーチの手順を簡略化したフローチャートを示します。

テーブル・サーチ手順は、物理アドレスを使用して変換テーブルにアクセスします。リード・モディファイ・ライト($\overline{\text{RMC}}$)信号がサーチの最初のバス・サイクルでアサートされ、最後までアサートされています。これによって、テーブル・サーチ全体が中断されずに終了することを保証しています。

サーチの最初のバス・サイクルでは、最初のテーブルのベース・アドレスとして適当なルート・ポインタのテーブル・アドレス・フィールドを使用します。アドレスの下位ビットは論理アドレスで与えられます。テーブルは、ファンクション・コードまたはTCレジスタのTIAフィールドで定義される論理アドレス・ビット群によってインデックスが付けられます。TCレジスタのFCLフィールドで、ファンクション・コードが使用されているかどうかを判断します。いずれの場合も、ルート・ポインタのディスクリプタ・タイプ・フィールドでインデックスのスケール・ファクタ(つまり乗数)を選択します。

最初のアクセスでディスクリプタを獲得します。ディスクリプタがテーブル・ディスクリプタの場合、MC68030は再びメモリをアクセスします。次のアクセスでは、ディスクリプタのテーブル・アドレスを、次のテーブルのベース・アドレスとして使用します。アドレスの下位ビットは論理アドレスで与えられます。テーブルは、ディスクリプタのディスクリプタ・タイプ・コードで決まるスケール・ファクタを用いて、論理アドレスからのビット群によってインデックスが付けられます。最初のテーブル・アクセスがファンクション・コードを使用している場合、2番目のアクセスではTCレジスタのTIAフィールドで選択されるビットを使用します。そうでない場合は、2番目のアクセスではTIBフィールドで選択されたビットを使用します。

アクセスでページ・ディスクリプタまたは無効ディスクリプタを取得するか、あるいはリミット違反が発生するまで、TIB、TIC、またはTIDの順に指定される論理アドレス・ビットを用いて、それ以降のアクセスを実行します。この時点で、アドレス・テーブルの全レベルがアクセスされたか否かにかかわらず、テーブル・サーチは終了します。ページ・ディスクリプタには、ATCエントリに必要な物理アドレスとその他の情報があります。MC68030はATCエントリを生成し、最初のバス・アクセスを再試行します。

図9-20にファンクション・コードおよび4つのTIxフィールドをすべて使用したテーブル・サーチを示します。

ロング・フォーマット・ディスクリプタが使用された場合、MC68030は次のレベルのテーブル・サーチに対するインデックス値に強制的にリミットを設定します。

ルート・ポインタには、ファンクション・コードのルックアップを使用しない(TCレジスタのFCLビットがゼロ)場合に適用されるリミット・フィールドがあります。次のレベルのテーブルにアクセ

表9-2　変換ツリーの選択

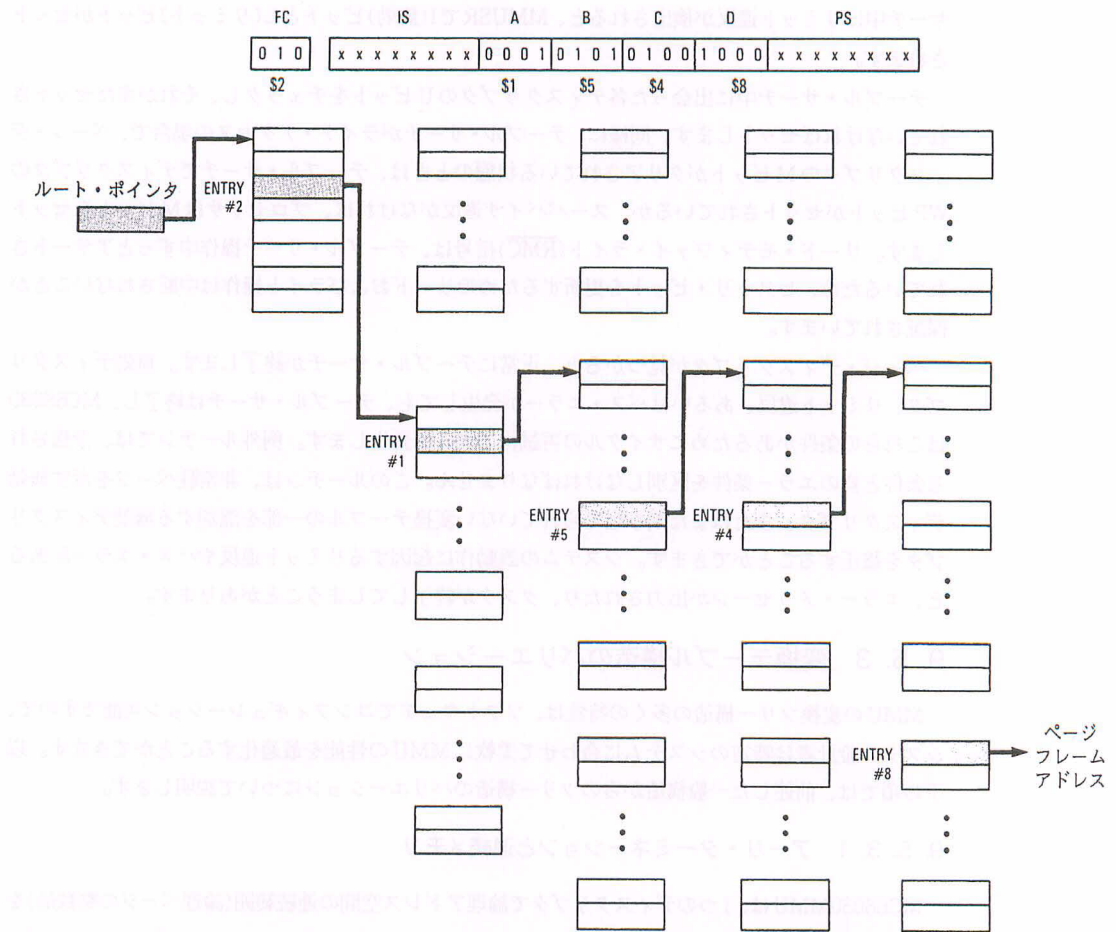
FC2	SRE	変換テーブルのルート・ポインタ
0	0	CRP
0	1	CRP
1	0	CRP
1	1	SRP

FCレベルのテーブル
Aレベル・テーブル（最大8テーブル、16エントリ/テーブル）
Bレベル・テーブル（最大128テーブル、16エントリ/テーブル）
Cレベル・テーブル（最大2Kテーブル、16エントリ/テーブル）
Dレベル・テーブル（最大32Kテーブル、16エントリ/テーブル）

図9-20　5レベルのテーブル・サーチ

スするのに使用するインデックスは、リミット・フィールドの内容と比較されます。リミット・フィールドはディスクリプタが適用されるアドレス空間の一部を効率的に低減し、あわせて変換テーブルのサイズも減少させます。インデックスは、リミット・フィールドで定義される範囲内になければなりません。リミットはL/Uビット値に基づいて、下限または上限のいずれかになります。L/Uビットがセットされているときには、リミットは下限となりこのリミットより小さいインデックスは境界外になります。

L/Uビットがゼロの場合、リミットは上限となりこのリミットより大きなインデックスは境界外になります。L/Uがセットされリミット・フィールドがゼロの場合、またはL/Uがクリアされリミット・フィールドが＄7FFFの場合、リミット・フィールドは効率的にディセーブルされます。

ノーマル変換またはPLOAD命令のテーブル・サーチ中に、リミット違反が検出されると、ATCにはB(バス・エラー)ビットがセットされたエントリがロードされます。PTEST命令のテーブル・

サーチ中にリミット違反が検出されると、MMUSRでI(無効)ビットとL(リミット)ビットがセットされます。

テーブル・サーチ中に出会った各ディスクリプタのUビットをチェックし、それがまだセットされていなければセットします。同様に、テーブル・サーチがライト・アクセスの場合で、ページ・ディスクリプタのMビットがクリアされている状態のときは、テーブル・サーチでディスクリプタのWPビットがセットされているか、スーパバイザ違反がなければ、プロセッサはMビットをセットします。リード・モディファイ・ライト($\overline{\text{RMC}}$)信号は、テーブル・サーチ操作中ずっとアサートされているため、ヒストリ・ビットを更新するためのリードおよびライト操作は中断されないことが保証されています。

ページ・ディスクリプタが見つかると、正常にテーブル・サーチが終了します。無効ディスクリプタ、リミット違反、あるいはバス・エラーが発生しても、テーブル・サーチは終了し、MC68030はこれらの条件があるためにサイクルの再試行で例外を発生します。例外ルーチンでは、予想される条件と真のエラー条件を区別しなければなりません。このルーチンは、非常駐ページを示す無効ディスクリプタ、またはまだ割り当てられていない変換テーブルの一部を識別する無効ディスクリプタを修正することができます。システムの誤動作に起因するリミット違反やバス・エラーがあると、エラー・メッセージが出力されたり、タスクが終了してしまうことがあります。

9. 5. 3 変換テーブル構造のバリエーション

MMUの変換ツリー構造の多くの特性は、ソフトウェアでコンフィギュレーション可能ですので、システム設計者は特定のシステムに合わせて柔軟にMMUの性能を最適化することができます。以下の項では、前述した一般構造からのツリー構造のバリエーションについて説明します。

9. 5. 3. 1 アーリ・ターミネーションと連続メモリ

MC68030MMUは、1つのディスクリプタで論理アドレス空間の連続範囲(論理ページの整数倍)を同等の連続物理アドレス範囲にマップすることができます。これは、ツリーで通常テーブル・ポインタをもつレベルに位置するディスクリプタのディスクリプタ・タイプ(DT)フィールドに“ページ・ディスクリプタ”コード($1)を設定して、そのテーブルのサブツリーを削除することにより行ないます。

サーチでページ・ディスクリプタが見つかると、それが変換ツリーの最下位レベルのページ・ディスクリプタ・テーブルにあるものでなくても、テーブル・サーチは終了します。

ポインタ・ディスクリプタ・テーブルにあるページ・ディスクリプタによってテーブル・サーチが終了すること(つまり、MC68030がゼロのTIxフィールドに出会わなかった)を、アーリ・ターミネーションと呼んでいます。また、終端にあるページ・ディスクリプタをアーリ・ターミネーション・ページ・ディスクリプタとよびます。

アーリ・ターミネーション・ページ・ディスクリプタは、変換テーブルで多数のページ・ディスクリプタの代わりをします。このディスクリプタは、それが置かれていたブランチ、およびそのブランチから出ているすべてのブランチに存在していたはずの全ページに適用されます。アーリ・ターミネーション・ページ・ディスクリプタは、物理メモリ内の連続ページが連続した論理ページに対応付けられているところで使用できます。アーリ・ターミネーション・ページ・ディスクリプタがロング・フォーマットの場合は、論理アドレスの次のインデックス・フィールドにリミット・フィールドが適用されます。これにより連続してマップされるページ数を制限することができます。詳細については、「9. 1. 2 変換テーブルのディスクリプタ」を参照してください。

ページ・ディスクリプタ・エンコーディングが見つかったときに、その論理ページ・アドレスの下位nビットが未使用の場合は、1つのディスクリプタがその論理ページ・アドレス(nの未使用ビッ

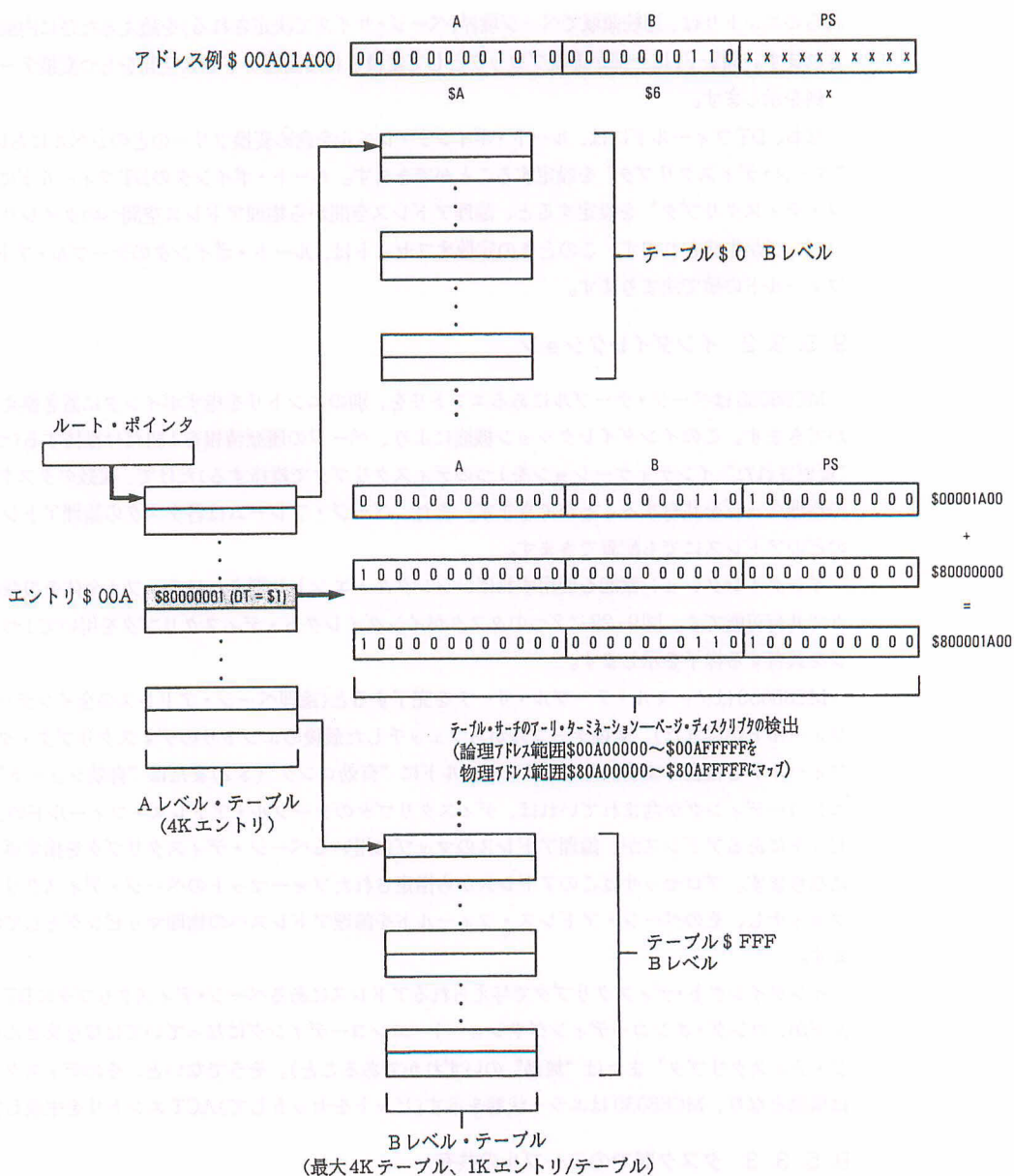



図9-21　連続メモリを使用した変換ツリーの例

トをゼロにセットして)から始まる論理アドレス空間の連続領域を、ページ・フレーム・ベース・アドレスから始まる2^{PS+n}バイトのサイズをもつ物理アドレス空間の連続領域にマップします。

アーリ・ターミネーション・ページ・ディスクリプタが適用される論理アドレスでサーチが行なわれると、MC68030はATCの中にその論理アドレスに対応するエントリを生成します。ATCエントリの物理アドレスは、ディスクリプタのページ・アドレス・フィールドにオフセットを加えたものになります。このオフセットは、サーチで使用したビットをゼロにセットした論理アドレスです。

アーリ・ターミネーション・ページ・ディスクリプタは、変換ツリーの中に連続領域のメンバをなす各ページについて個別にディスクリプタを維持しなくても、連続した論理-物理マッピングを生成することができますが、ATCはマップされている各ページにつき1つのエントリを保持します。こ

れらのエントリは、連続領域でページ境界(ページ・サイズで決定される)を超えるたびに内部で生成されます。図9-21に一部が連続ブロックとして変換される論理アドレス空間をもつ変換テーブルの一例を示します。

なお、DTフィールドには、ルート・ポインタ・レベルを含め変換ツリーのどのレベルにおいても、“ページ・ディスクリプタ”を設定することができます。ルート・ポインタのDTフィールドに“ページ・ディスクリプタ”を設定すると、論理アドレス空間から物理アドレス空間へのダイレクト・マッピングが生成されます。このときの定数オフセットは、ルート・ポインタのテーブル・アドレス・フィールドの値で決まります。

9. 5. 3. 2 インダイレクション

MC68030はページ・テーブルにあるエントリを、別のエントリを指すポインタに置き換えることができます。このインダイレクション機能により、ページの履歴情報を1組だけ維持する(つまり、“変更された”インディケーションを1つのディスクリプタで維持する)だけで、複数のタスクが1つの物理ページを共有することができます。また、ページ・フレームは各タスクの論理アドレス空間のどのアドレスにでも配置できます。

インダイレクション機能を使用すれば、シングル・エントリ群またはテーブル全体を複数のタスクで共有可能です。図9-22に2つのタスクがインダイレクト・ディスクリプタを用いて1つのページを共有する様子を示します。

MC68030はノーマル・テーブル・サーチを完了すると(論理ページ・アドレスの全インデックス・フィールドが尽きた)、変換テーブルからフェッチした最後のエントリのディスクリプタ・タイプ・フィールドを検査します。そのDTフィールドに“有効ロング”($2)または“有効ショート”($3)エンコーディングが含まれていれば、ディスクリプタのテーブル・アドレス・フィールドの上位30ビットにあるアドレスが、論理アドレスのマップに用いるページ・ディスクリプタを指すポインタになります。プロセッサはこのアドレスから指定されたフォーマットのページ・ディスクリプタをフェッチし、そのページ・アドレス・フィールドを論理アドレスへの物理マッピングとして使用します。

インダイレクト・ディスクリプタで与えられるアドレスにあるページ・ディスクリプタのDTフィールドが、ロング・エンコーディングやショート・エンコーディングになっていてはなりません(“ページ・ディスクリプタ”または“無効”のいずれかであること)。そうでないと、そのディスクリプタは無効となり、MC68030はエラー状態を示す(ビットをセットして)ACTエントリを生成します。

9. 5. 3. 3 タスク間でのテーブルの共有

ページ・テーブルまたはポインタ・テーブルは、複数のタスクのアドレス変換テーブルの共有テーブルを指すポインタを置けば、タスク間でページまたはポインタ・テーブルを共有できます。上位(非共有)テーブルでは、保護ビットをいくとおりにも設定できるため、いろいろなタスクが異なるパーミッションをもつエリアを使用することがきるのです。図9-23では、2つのタスクがBレベルにあるテーブルで変換されるメモリを共有しています。なお、タスク“A”は共有エリアに書込みを行うことはできません。しかし、タスク“B”は、共有テーブルを指すポインタのWPビットがクリアされていますので、共有エリアの読出しおよび書込みを行なうことができます。また、この共有エリアは各タスクのさまざまな論理アドレスに現われます。

9. 5. 3. 4 テーブルのページング

アクティブなタスクのアドレス変換ツリー全体が、一度にメイン・メモリ内に存在する必要はありません。つまり、ページのワーキング・セットだけがメイン・メモリにあればよく、ページの常

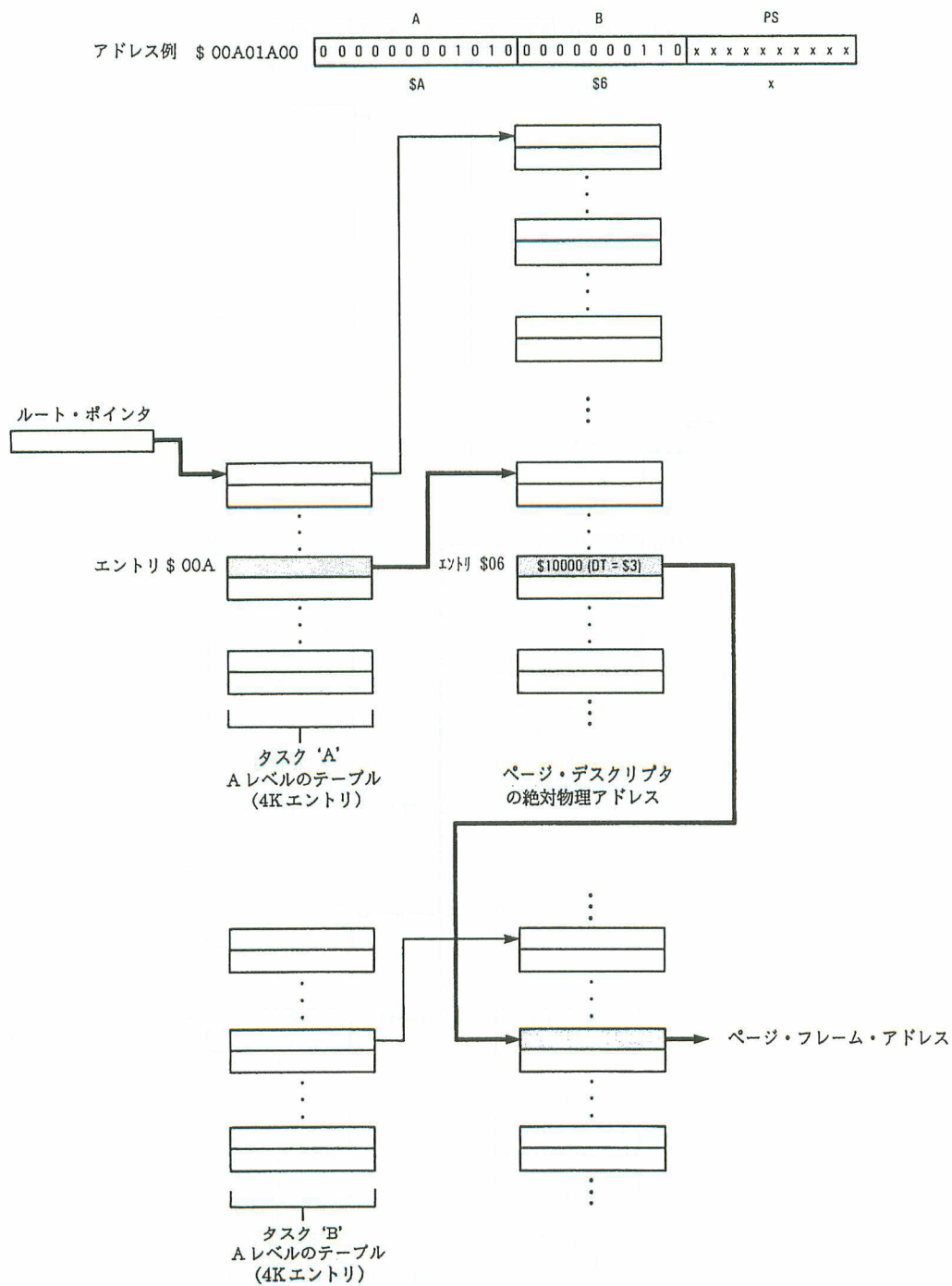



図9-22　インダイレクト・ディスクリプタを使用した変換ツリーの例

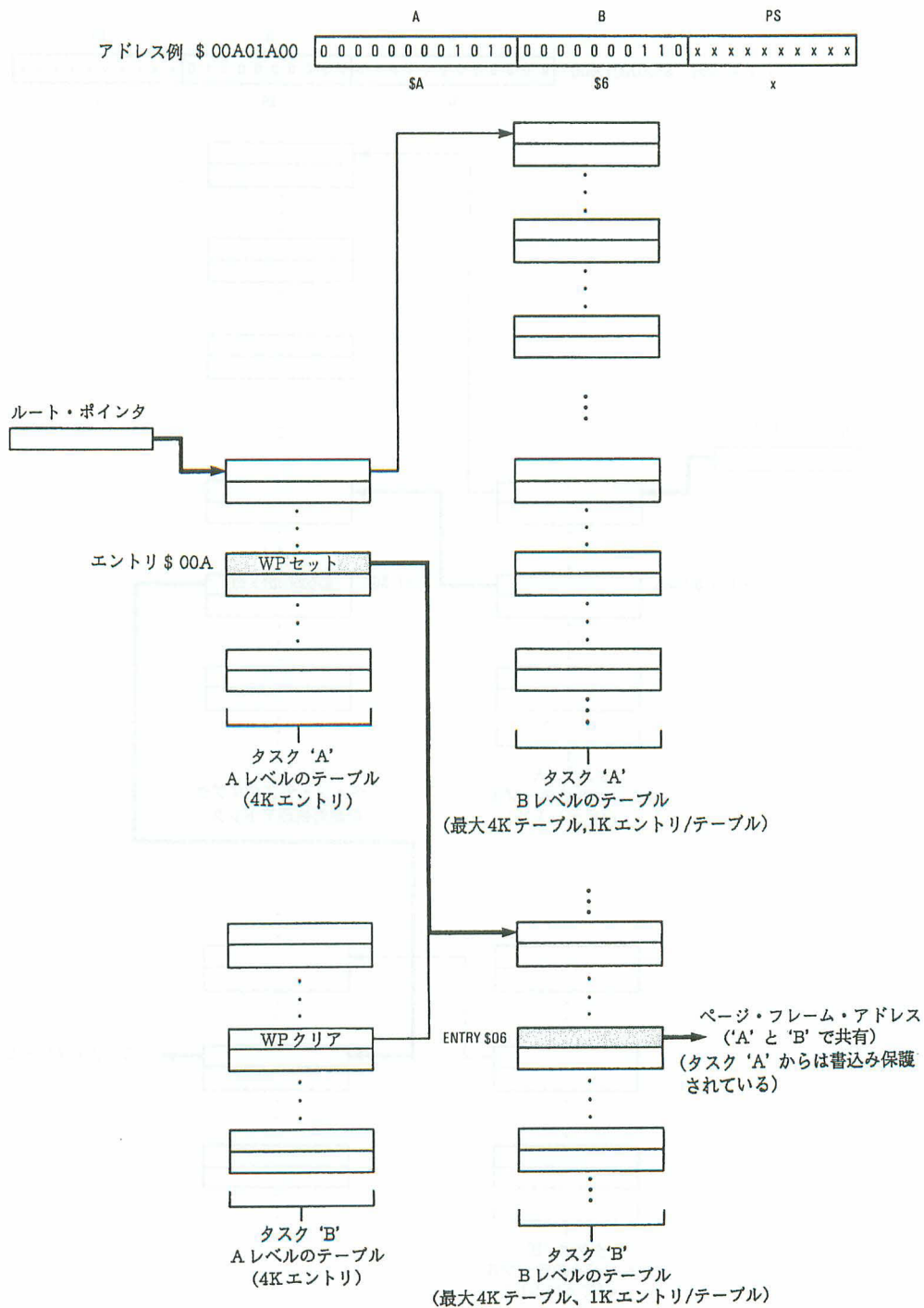



図9-23　共有テーブルを使用した変換ツリーの例

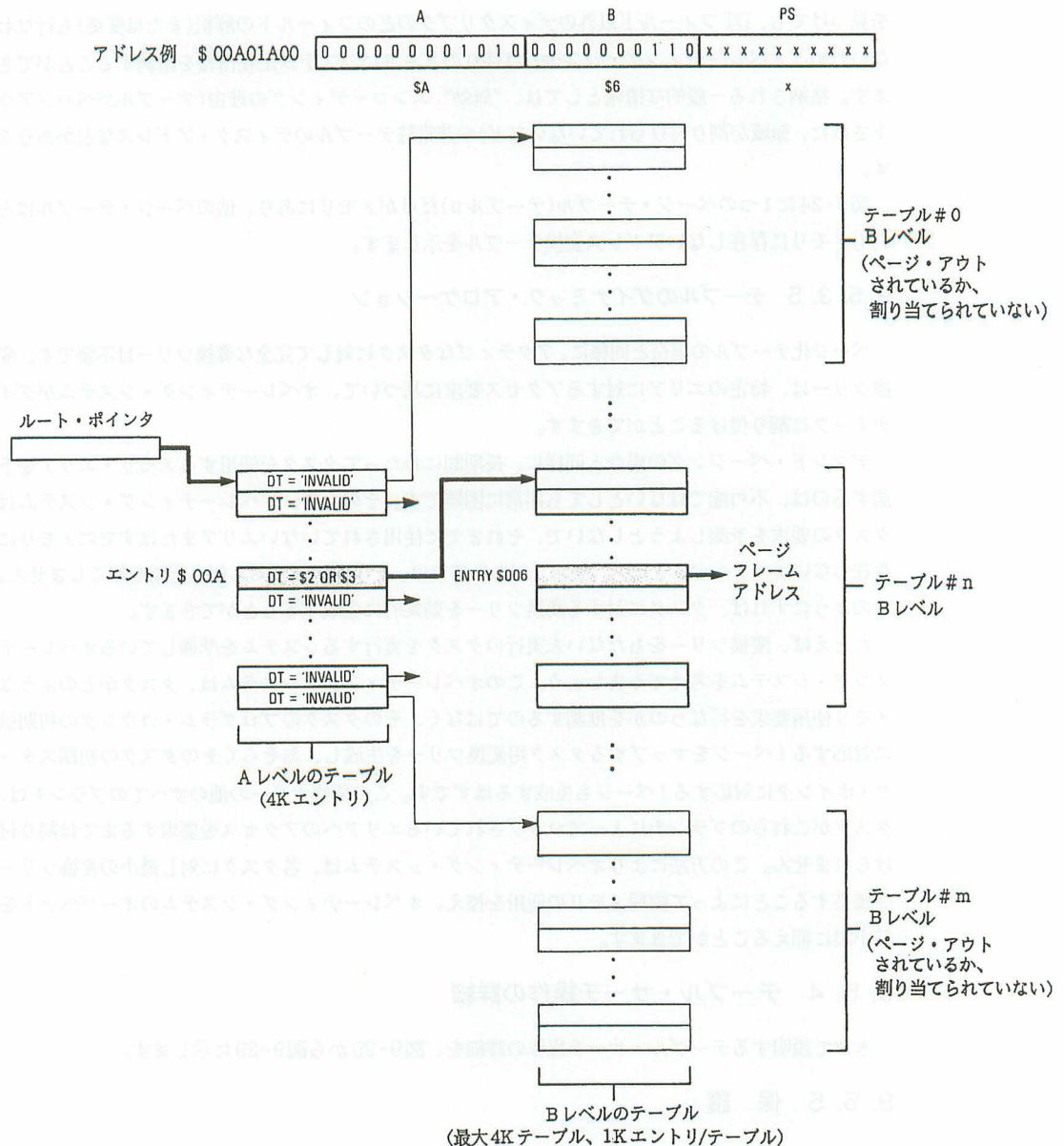



図9-24　非常駐テーブルのある変換ツリーの例

駐セットを記述するテーブルだけがメイン・メモリに入っていることが必要です。テーブルのページングは、不在テーブルを指すテーブル・ディスクリプタのDTフィールドに“無効”コード($0)を設定すれば実現できます。タスクが不在テーブルで変換されるはずであったアドレスを使用しようとすると、MC68030は変換ツリーをたどることができず、実行ユニットがテーブル・サーチを開始させたバス・サイクルをリトライするときに、バス・エラー例外を受け取ります。

非常駐テーブルに対応するディスクリプタの“無効”コードを確認するのはシステム・ソフトウェアの責任です。これは、ディスクリプタの“未使用ビット”を使用して、“無効”エンコーディングに関係するステータス情報を格納すれば簡単に行なえます。MC68030が“無効”ディスクリプタ

を見つけても、DTフィールド以外のディスクリプタのどのフィールドの解釈(または変更)も行なわないため、オペレーティング・システムは残りのビットにシステム定義情報を格納することができます。格納される一般的な情報としては、"無効"エンコーディングの理由(テーブルがページアウトされた、領域が割り付けられていないなど)や非常駐テーブルのディスク・アドレスなどがあります。

図9-24に1つのページ・テーブル(テーブルn)だけがメモリにあり、他のページ・テーブルはどれもメモリに存在しないアドレス変換テーブルを示します。

9. 5. 3. 5　テーブルのダイナミック・アロケーション

ページ化テーブルの場合と同様に、アクティブなタスクに対して完全な変換ツリーは不要です。変換ツリーは、特定のエリアに対するアクセス要求に基づいて、オペレーティング・システムがダイナミックに割り付けることができます。

デマンド・ページングの場合と同様に、長期間にわたってタスクが使用するメモリ・エリアを予測するのは、不可能ではないとしても非常に困難です。そのため、オペレーティング・システムはタスクの要求を予測しようとしないで、それまでに使用されていないエリアまたはすでにメモリに存在しないエリアへのアクセス"要求"があるまでは、そのタスクに対し何も行動を起こしません。このようにすれば、タスクに対する変換ツリーを効果的に生成することができます。

たとえば、変換ツリーをもたない未実行のタスクを実行するシステムを準備しているオペレーティング・システムを考えてみましょう。このオペレーティング・システムは、タスクがどのようなメモリ使用要求を行なうのかを推測するのではなく、そのタスクのプログラム・カウンタの初期値に対応する1ページをマップするタスク用変換ツリーを生成し、おそらくそのタスクの初期スタック・ポインタに対応する1ページも生成するはずです。この変換ツリーの他のすべてのブランチは、タスクがこれらのブランチによってマップされているエリアへのアクセスを要求するまでは割り付けられません。この方法によりオペレーティング・システムは、各タスクに対し最小の変換ツリーを構築することによって物理メモリの使用を控え、オペレーティング・システムのオーバヘッドを最小限に抑えることができます。

9. 5. 4　テーブル・サーチ操作の詳細

本章で説明するテーブル・サーチ操作の詳細を、図9-25から図9-29に示します。

9. 5. 5　保　護

M68000ファミリのプロセッサはファンクション・コード信号によりサイクル単位で動作するコンテクストを表示します。これらの信号はユーザ・プログラム空間、ユーザ・データ空間、スーパバイザ・プログラム空間、およびスーパバイザ・データ空間へのアクセスを識別します。変換制御(TC)レジスタのファンクション・コード・ルックアップ(FCL)ビットをセットすることにより、ファンクション・コード信号を保護メカニズムに使用することができます。

MC68030 MMUは使用するスーパバイザ空間およびユーザ空間に対して別々の変換ツリーを用意することができます。TCレジスタのスーパバイザ・ルート・ポインタ・イネーブル・ビット(SRE)がセットされているときは、スーパバイザ・ルート・ポインタがスーパバイザ空間用変換ツリーに対するルート・ポインタ・レジスタとして使用されます。

変換テーブル・ツリーにはマッピングおよび保護情報の両方が含まれています。各テーブルおよびページ・ディスクリプタには、どのレベルでもセットして書込み保護を行なうことが可能な書込み保護(WP)ビットがあります。各ロング・フォーマット・テーブルおよびページ・ディスクリプタにも、スーパバイザ専用(S)ビットがあり、スーパバイザ特権レベルで動作しているプログラムへの

FC2	SRE	ROOT
0	0	CRP
0	1	CRP
1	0	CRP
1	1	SRP

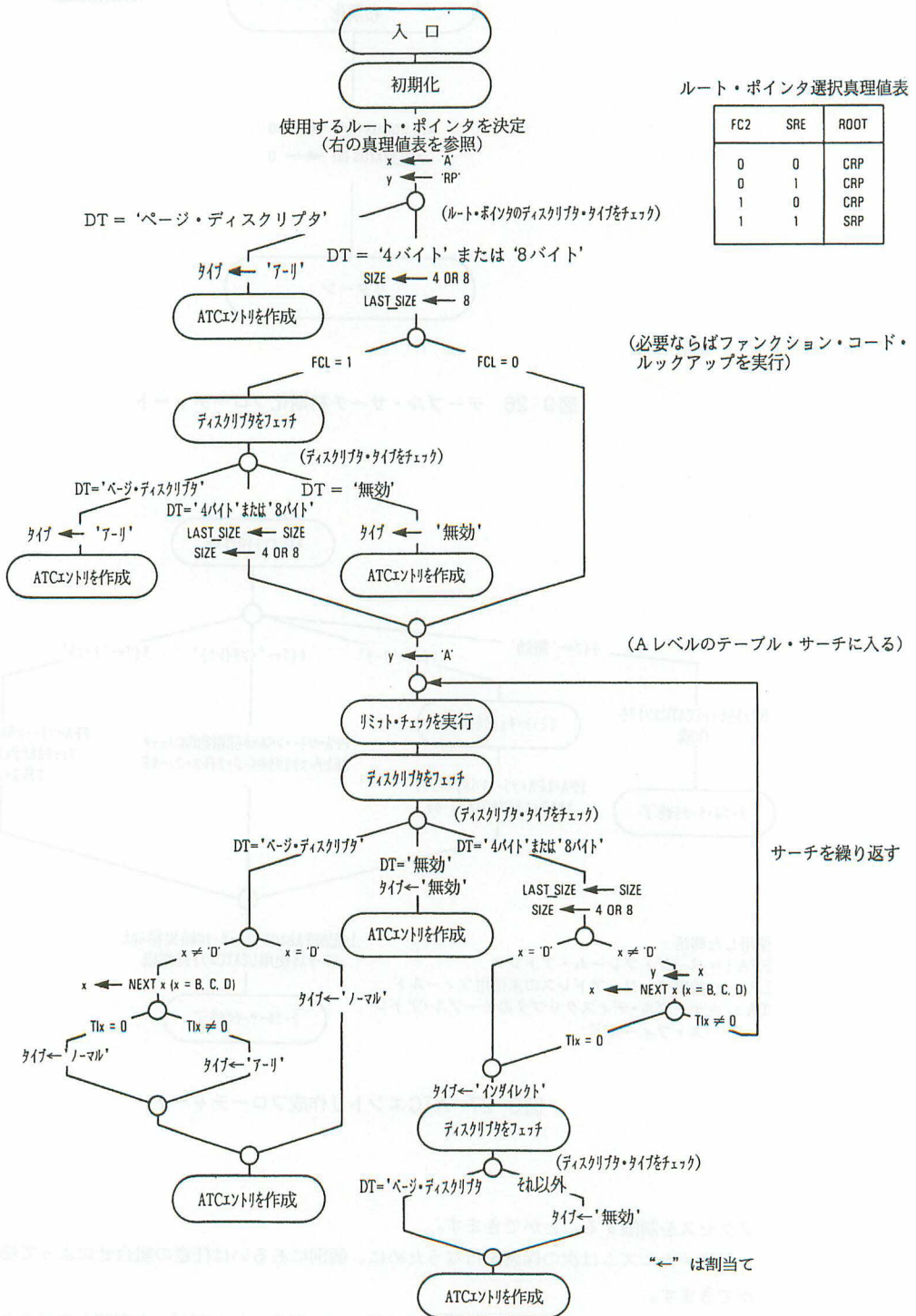



図9-25　MMUテーブル・サーチ操作の詳細フローチャート

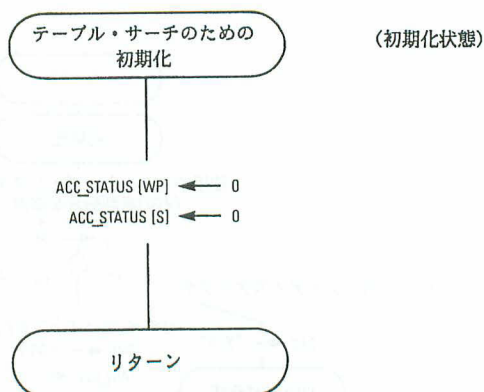



図9-26　テーブル・サーチ初期化フローチャート

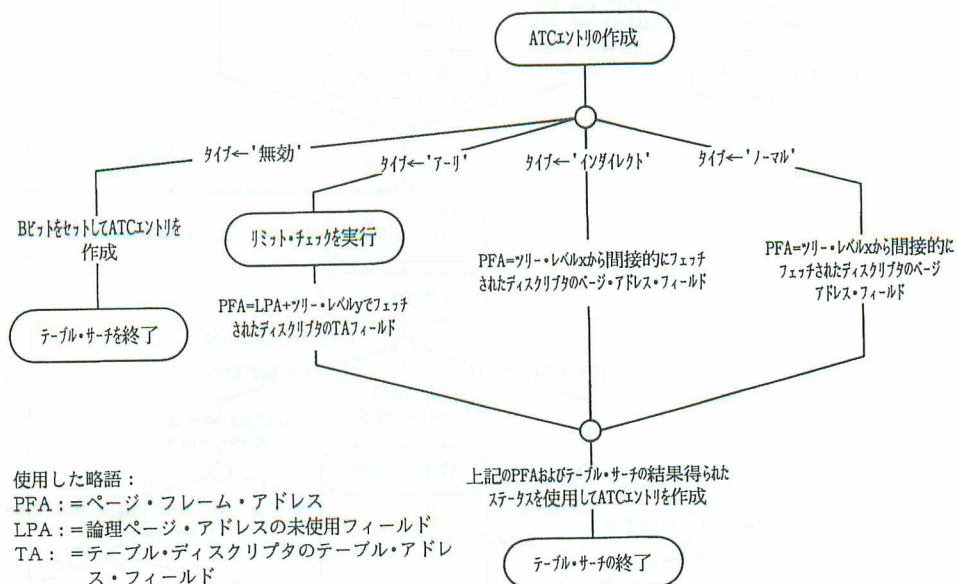



図9-27　ATCエントリ作成フローチャート

アクセスを制限することができます。

保護メカニズムは次の保護を行なうために、個別にあるいは任意の組合せによって使用することができます。

- ユーザ・プログラムからのスーパバイザ・プログラムおよびデータ空間へのアクセス
- 他のユーザ・プログラムまたはスーパバイザ・プログラムからのユーザ・プログラムおよびデータ空間へのアクセス(MOVES命令によるアクセスを除く)
- スーパバイザおよびユーザ・プログラム空間へのライト・アクセス(MOVES命令を使用したスーパバイザによるアクセスを除く)

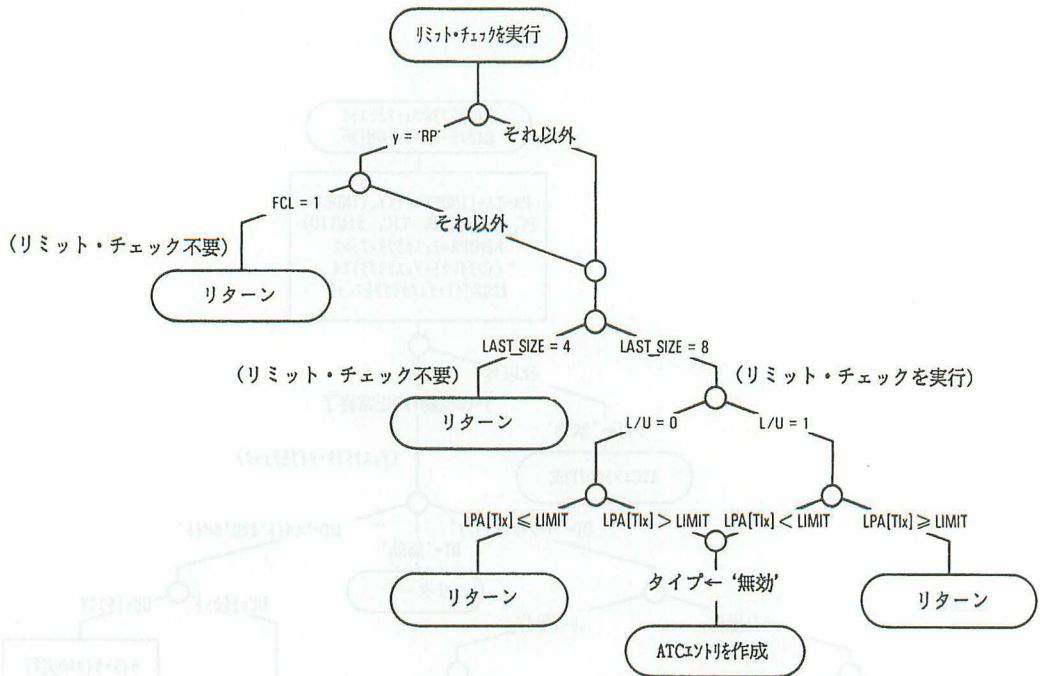



図9-28　リミット・チェック手順のフローチャート

●単一または複数のメモリ・ページへのライト・アクセス

9. 5. 5. 1　ファンクション・コードのルックアップ

許可されないアクセスからスーパバイザおよびユーザ空間を保護する1つの方法は、TCレジスタのFCLビットをセットすることです。これにより、論理アドレス空間を図9-30に示すように、効果的にスーパバイザ・プログラム空間、スーパバイザ・データ空間、ユーザ・プログラム空間、およびユーザ・データ空間にセグメント化します。各タスクは、ユーザ空間の論理アドレスに対する固有のマッピングを備えたアドレス変換ツリーをもっています。スーパバイザ空間をマッピングするための変換テーブルは、各タスクの変換ツリーにコピーすることができます。。図9-31にファンクション・コードのルックアップを用いた変換ツリーを示し、図9-32に共通のスーパバイザ空間を共有する2つのタスクに対する変換ツリーを示します。

9. 5. 5. 2　スーパバイザ変換ツリー

第2の保護メカニズムはスーパバイザ変換ツリーを使用します。スーパバイザ変換ツリーは、ユーザ・プログラムによるアクセスからスーパバイザ・プログラムおよびデータを保護し、そしてスーパバイザ・プログラムによるアクセスからユーザ・プログラムおよびデータを保護します。アクセスは「アドレス空間の転送(MOVES)」命令でメモリのどの領域にでもアクセスできるスーパバイザ・プログラムに許可されます。

TCレジスタのSREビットがセットされていると、スーパバイザ・レベルのすべてのアクセスに対して、SRPで指し示される変換ツリーを選択します。この変換ツリーは全タスクに共通にすることができます。この手法は、変換ツリーにファンクション・コード・レベルを追加することなく、論理アドレス空間をユーザおよびスーパバイザ領域にセグメント化します。

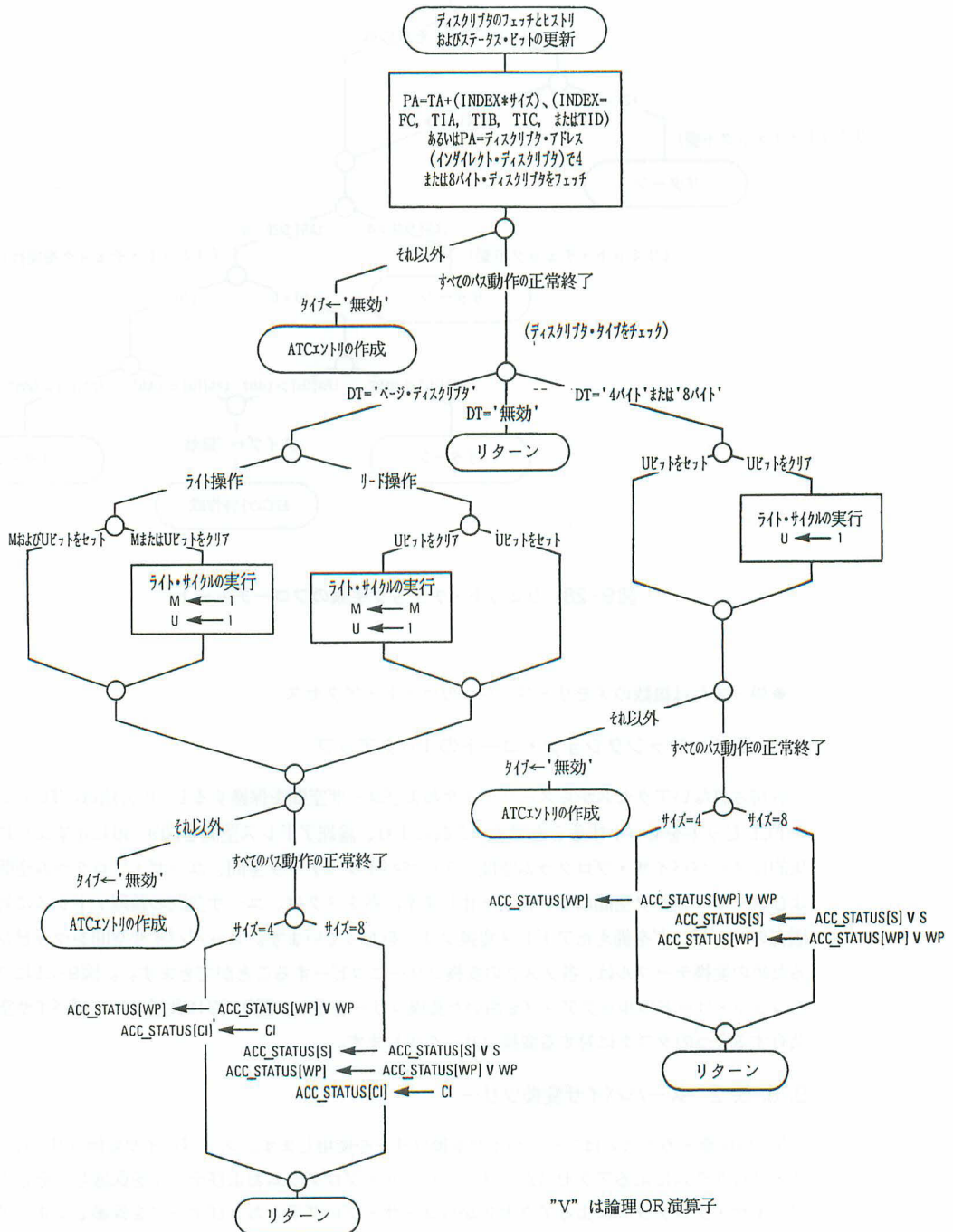



図9-29　ディスクリプタ・フェッチ操作の詳細フローチャート

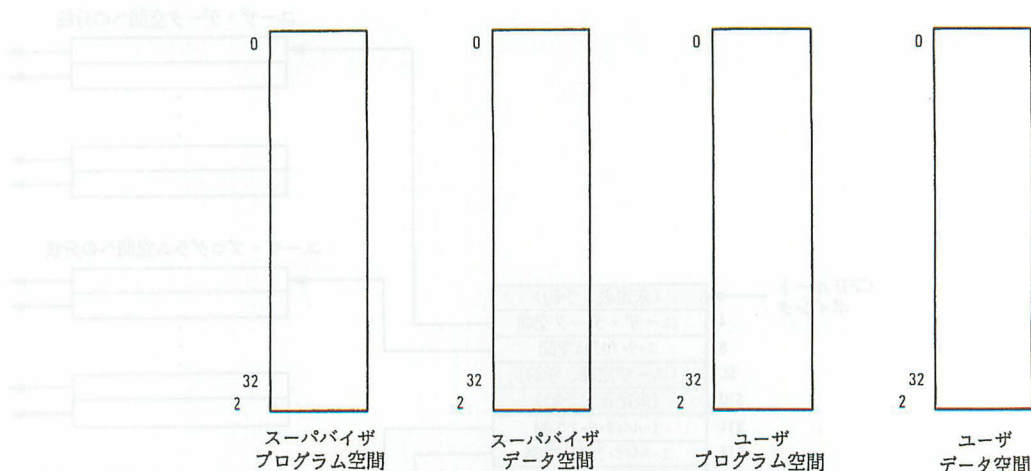



図9-30　ファンクション・コード・ルックアップを使用した論理アドレス・マップ

9.5.5.3　スーパバイザ専用

第3のメカニズムは、論理アドレス空間をスーパバイザおよびユーザ・アドレス空間にセグメント化することなく、スーパバイザ・プログラムおよびデータを保護します。ロング・フォーマットのテーブル・ディスクリプタおよびページ・ディスクリプタには、ユーザ・プログラムによるアクセスからメモリの領域を保護するためのSビットが含まれています。ユーザ・アクセスに対するテーブル・サーチで、いずれかのテーブルまたはページ・ディスクリプタのSビットがセットされているのが見つかると、テーブル・サーチを終了して、論理アドレスに対応するATCディスクリプタとBビットをセットして生成します。それ以降ユーザ・アクセスの再試行を行なうと、バス・エラー例外になります。Sビットを使用すれば、ユーザ・プログラムのアクセスから変換ツリーのブランチで定義されたメモリ全域、あるいは単一または複数のページを保護することができます。

9.5.5.4　書込み保護

MC68030はプログラムおよびデータに対してセグメント化されたアドレス空間に個々に書込み保護を与えます。テーブルおよびページ・ディスクリプタはすべて、任意の種類のライト・アクセスからメモリの領域を保護するためのWPビットをもっています。テーブル・サーチで、どれかのテーブルまたはページ・ディスクリプタのWPビットがセットされているのが見つかると、テーブル・サーチを終了し、その論理アドレスに対応するATCディスクリプタをWPビットをセットして生成します。それ以降ライト・アクセスの再試行を行なうと、バス・エラー例外になります。WPビットを使用すれば、ライト・アクセスから変換ツリーのブランチで定義されたメモリ全域、あるいは単一または複数のページを保護することができます。図9-33に、保護を行なうためのスーパバイザ専用および書込み保護ビットを使用するために編成した論理アドレス空間のメモリ・マップを示します。また、図9-34にこの手法のための変換例を示します。

9.6　MC68030およびMC68851のMMUの相違点

MC68851ページ式メモリ管理ユニットは、MC68020にコプロセッサとしてメモリ管理機能を提

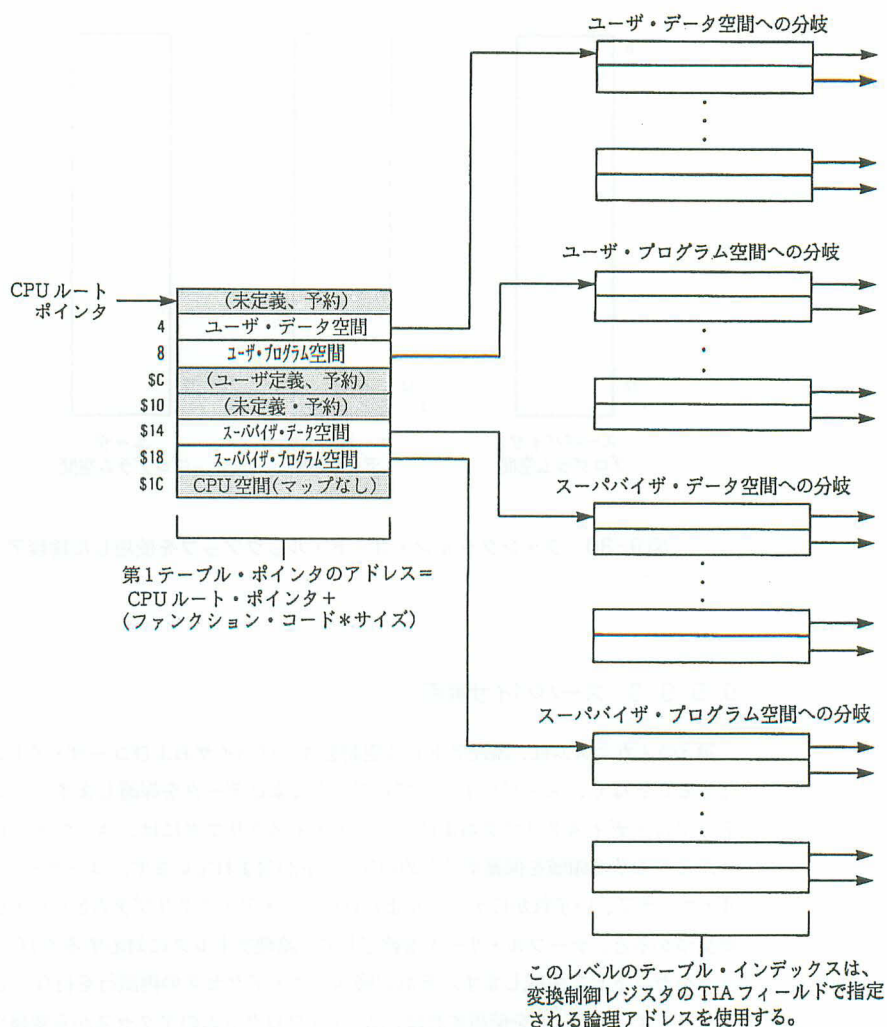



図9-31　ファンクション・コード・ルックアップを用いた変換ツリーの例

供します。MC68030のオンチップMMUは、MC68020/MC68851ペアのもつ多くの特長を備えていますが、MC68851の次の機能は、MC68030MMUでは使用できません。

- ●アクセス・レベル
- ●ブレークポイント・レジスタ
- ●ルート・ポインタ・テーブル
- ●タスクの別名付け
- ●ATCでのロック可能エントリ
- ●グローバル共有として定義するATCエントリ

また、次のMC68030の特長はMC68020/MC68851ペアとは異なります。

- ●22エントリのATC
- ●命令セットの削減
- ●MMU命令では制御可変アドレッシング・モードのみサポート

一般に、MC68030はMC68020/MC68851の組合せとプログラムに互換性があります。しかし、

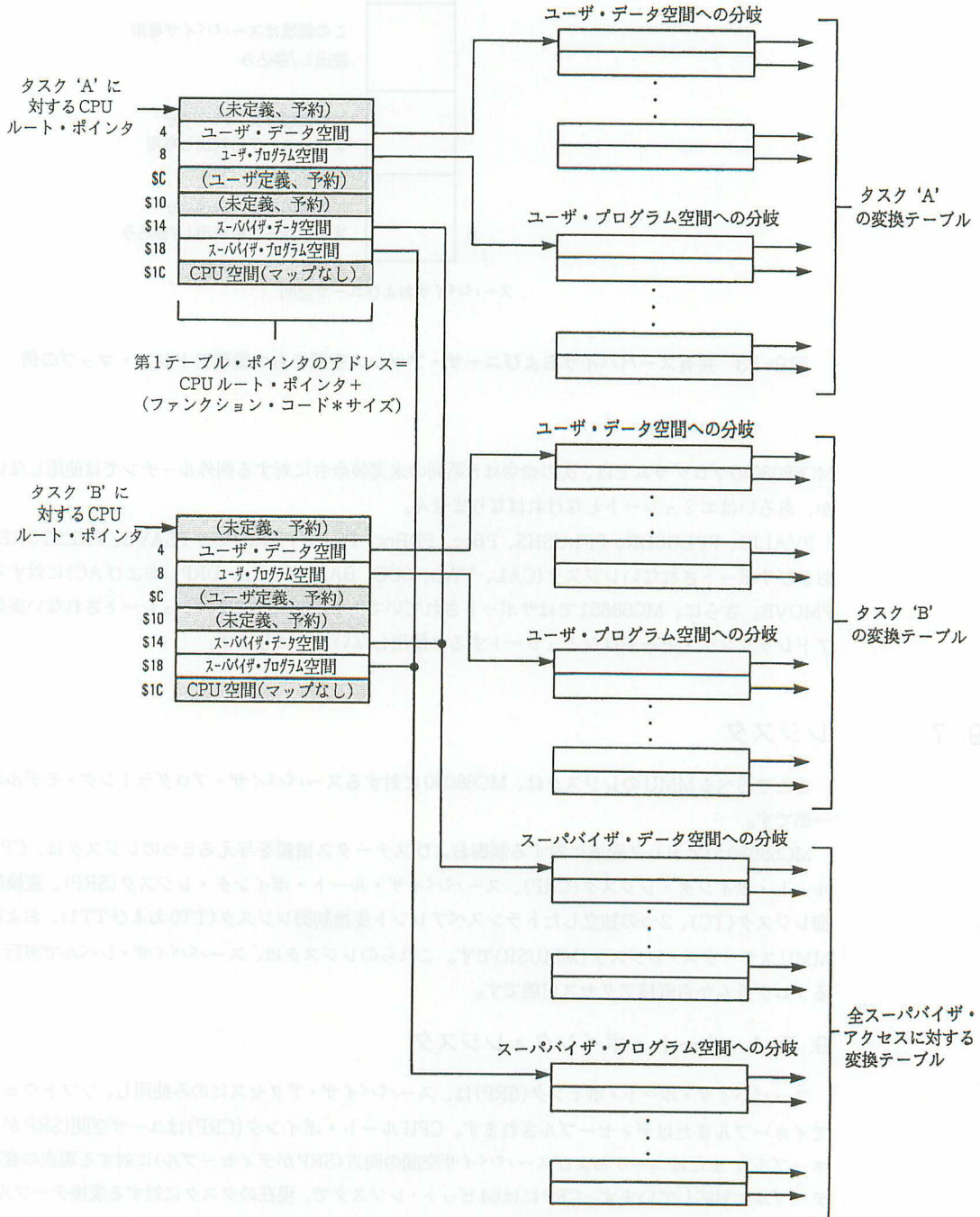



図9-32　2タスクに対する変換ツリー構造の例

スーパバイザおよびユーザ空間

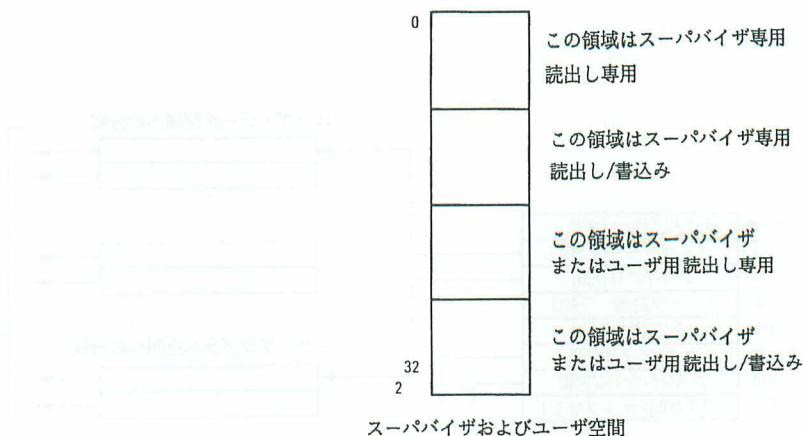



図9-33　共有スーパバイザおよびユーザ・アドレス空間をもつ論理アドレス・マップの例

MC68030のプログラムでは、次の命令はF系列の未実装命令に対する例外ルーチンでは使用しないか、あるいはエミュレートしなければなりません。

PVALID、PFLUSHR、PFLUSHS、PBcc、PDBcc、PScc、PTRAPcc、PSAVE、PRESTORE、およびサポートされないレジスタ(CAL、VAL、SCC、BAD、BACx、DRP、およびAC)に対するPMOVE。さらに、MC68851ではサポートされていてもMC68030ではエミュレートされない実効アドレッシング・モードはシミュレートするか使用しないようにします。

9. 7　レジスタ

ここで述べるMMUのレジスタは、MC68030に対するスーパバイザ・プログラミング・モデルの一部です。

MC68030のアドレス変換に対する制御およびステータス情報を与える6つのレジスタは、CPUルート・ポインタ・レジスタ(CRP)、スーパバイザ・ルート・ポインタ・レジスタ(SRP)、変換制御レジスタ(TC)、2つの独立したトランスペアレント変換制御レジスタ(TT0およびTT1)、およびMMUステータス・レジスタ(MMUSR)です。これらのレジスタは、スーパバイザ・レベルで実行するプログラムから直接アクセス可能です。

9. 7. 1　ルート・ポインタ・レジスタ

スーパバイザ・ルート・ポインタ(SRP)は、スーパバイザ・アクセスにのみ使用し、ソフトウェアでイネーブルまたはディセーブルされます。CPUルート・ポインタ(CRP)はユーザ空間(SRPがイネーブル)、またはユーザおよびスーパバイザ空間の両方(SRPがディセーブル)に対する現在の変換テーブルに対応しています。CRPには64ビット・レジスタで、現在のタスクに対する変換テーブル・ツリーのルートのアドレスおよび関連ステータス情報が含まれています。新しいタスクの実行を開始すると、オペレーティング・システムは、通常CRPに新しいルート・ポインタ・ディスクリプタを書き込みます。新しい変換テーブル・アドレスは、アドレス変換キャッシュ(ATC)の内容がすでに有効ではないことを意味します。したがって、CRPをロードする命令はオプションでATCをフラッシュすることができます。

SRPは64ビット・レジスタであり、オプションでスーパバイザ領域にアクセスするための変換テー

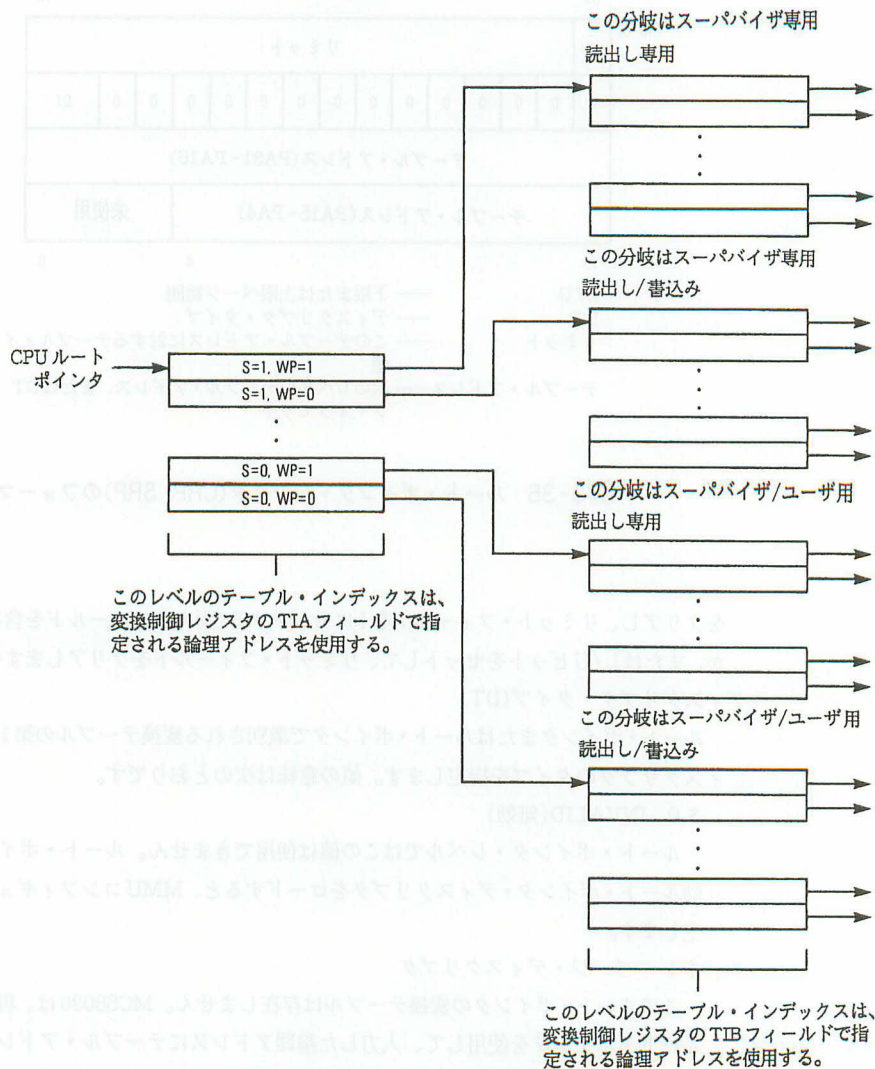



図9-34　SおよびWPビットを使用して保護を設定する変換ツリーの例

ブルのルートのアドレスおよび関連ステータス情報を保持しています。SRPはスーパバイザ特権レベルで動作しているときに、変換制御レジスタ(TC)のスーパバイザ・ルート・ポインタ・イネーブル・ビット(SRE)がセットされた場合にのみ使用されます。SRPをロードする命令は、オプションでATCをフラッシュできます。CRPおよびSRPのフォーマットは図9-35に示すとおりです。次のフィールドが定義されています。

上限/下限(L/U)

　このビットがセットされたときは、リミット・フィールド値を、変換テーブル・インデックスの符号なし下限値として使用するよう指定します。このビットがクリアされているときには、リミット・フィールドは変換テーブル・インデックスの符号なし上限値です。

リミット

　テーブル・サーチの次のレベルで使用するインデックスに対する最大値および最小値を指定します(ファンクション・コード・レベルは制限されない)。制限機能を禁止するには、L/Uビット

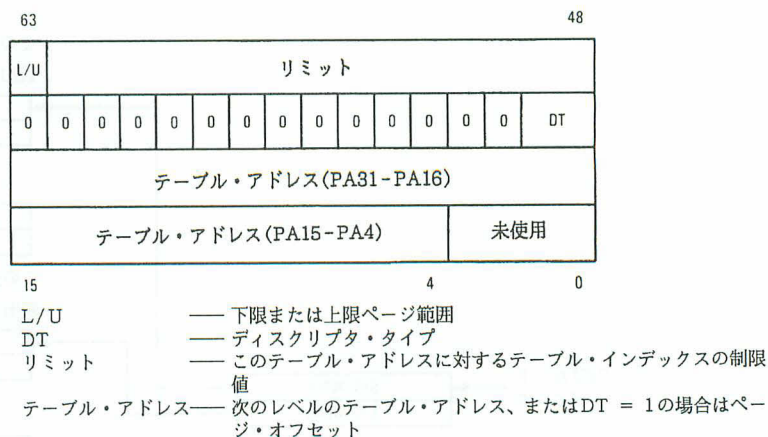



図9-35 ルート・ポインタ・レジスタ(CRP, SRP)のフォーマット

をクリアし、リミット・フィールドを1にセットする(両方のフィールドを含むワードが$7FFF)か、またはL/Uビットをセットして、リミット・フィールドをクリアします("$8000"を設定)。

ディスクリプタ・タイプ(DT)

ルート・ポインタまたはルート・ポインタで識別される変換テーブルの第1レベルに含まれるディスクリプタのタイプを指定します。値の意味は次のとおりです。

$0 INVALID(無効)

ルート・ポインタ・レベルではこの値は使用できません。ルート・ポインタ・レジスタに無効ルート・ポインタ・ディスクリプタをロードすると、MMUコンフィギュレーション例外が発生します。

$1 ページ・ディスクリプタ

このルート・ポインタの変換テーブルは存在しません。MC68030は、現在のページ内のこのルート・ポインタを使用して、入力した論理アドレスにテーブル・アドレス・フィールドの値を加算(符号なし)して、アクセスに対するATCエントリ(ページ・ディスクリプタ)を内部で計算します。結果的に、一定のオフセット(テーブル・アドレス)をもつ直接マッピングを行ないます。この場合、プロセッサはTCレジスタのFCLビットの状態に関係なくリミット・チェック

図9-36 変換制御レジスタ(TC)のフォーマット

を実行します。

$2 VALID(有効)4バイト

変換ツリーのルートにある変換テーブルには、ショート・フォーマットのディスクリプタがあります。MC68030は次のディスクリプタをアクセスするために、このレベルのテーブル・サーチに対してテーブル・インデックスを4バイトでスケールしなければなりません。

$3 VALID(有効)8バイト

変換ツリーのルートにある変換テーブルには、ロング・フォーマットのディスクリプタがあります。MC68030は次のディスクリプタをアクセスするために、このレベルのテーブル・サーチに対してテーブル・インデックスを8バイトでスケールしなければなりません。

テーブル・アドレス

ルート・ポインタ・レベルにある変換テーブルの物理ベース・アドレス(ビット4-31)が含まれています。DTフィールドに$1があるときは、テーブル・アドレス・フィールドの値をオフセットとして使用して、ページ・ディスクリプタの物理アドレスを計算します。テーブル・アドレス・フィールドにはゼロが含まれている場合もあります(ゼロ・オフセットのとき)。

未使用

ルート・ポインタのビット0-3は使用されておらず、書込みを行なうと無視されます。他のすべての未使用ビットは常にゼロでなければなりません。

9.7.2　変換制御レジスタ

変換制御レジスタ(TC)は32ビット・レジスタで、アドレス変換のための制御フィールドがあります。このレジスタの未実装フィールドはすべてゼロで読み出されます。また、常にゼロで書込みを行なわなければなりません。

このレジスタに書き込むことにより、オプションで全体のATCをフラッシュさせることができます。Eビット(ビット31)をセット(変換イネーブル)して書込みを行なうと、次のようにPS、IS、およびTIxの値について一貫性チェックが行なわれます。TIxフィールドは、ゼロ・フィールドに達するまで加えられ、この和がPSおよびISに加算されます。合計は32でなければなりません。そうでない場合は、MMUコンフィギュレーション例外(「9.7.5.3　MMUコンフィギュレーション例外」参照)が発生します。MMUコンフィギュレーション例外が発生すると、そのデータでTCレジスタが更新され、Eビットがクリアされます。変換制御レジスタを図9-36に示します。

これらのフィールドを説明します。

イネーブル(E)

このビットはアドレス変換を次のとおりイネーブルおよびディセーブルします。

0－変換をディセーブル

1－変換をイネーブル

このビットはリセットによりクリアされます。変換がディセーブルされていると、論理アドレスを物理アドレスとして使用します。Eビットの状態に関係なく、MMU命令(PTEST、PLOAD、PMOVE、PFLUSH)を正常に使用できます。また、Eビットがセットされていても、Eビットがセットされた値でTCレジスタを更新することができます。Eビットの状態はトランスペアレント変換レジスタの使用には影響を与えません。

スーパバイザ・ルート・ポインタ・イネーブル(SRE)

このビットはスーパバイザ・ルート・ポインタ・レジスタ(SRP)の使用を制御します。

0－SRPをディセーブル

1－SRPをイネーブル

SRPがディセーブルされたときは、ユーザおよびスーパバイザ・アクセスは、CRPで定義され

る変換テーブルを使用します。SRPがイネーブルされると、ユーザ・アクセスはCRPを使用し、スーパバイザ領域アクセスはSRPを使用します。

ファンクション・コード・ルックアップ(FCL)

このビットは次のとおり、アドレス変換テーブルをサーチするためのファンクション・コード・ルックアップの使用をイネーブルします。

0－ファンクション・コード・ルックアップ・ディセーブル

1－ファンクション・コード・ルックアップ・イネーブル

ファンクション・コード・ルックアップがディセーブルされると、変換テーブル構造内の第1レベルのポインタ・テーブルが、TIAで定義される論理アドレス・フィールドによってインデックスが付けられます。ファンクション・コード・ルックアップがイネーブルされると、変換テーブル構造の第1テーブルがファンクション・コードによってインデックスが付けられます。この場合、CRPまたはSRPのリミット・フィールドは無視されます。

ページ・サイズ(PS)

この4ビット・フィールドでシステム・ページのサイズを指定します。

1000 － 256バイト

1001 － 512バイト

1010 － 1Kバイト

1011 － 2Kバイト

1100 － 4Kバイト

1101 － 8Kバイト

1110 － 16Kバイト

1111 － 32Kバイト

これ以外のすべてのビットの組合せは、将来の使用のためにモトローラにより予約されています。TCレジスタのこのフィールドに、上記以外の値をロードしようとすると、MMUコンフィギュレーション例外が発生します。

イニシャル・シフト(IS)

この4ビット・フィールドは、テーブル・サーチ中に無視される論理アドレスの上位ビットを保持します。このフィールドには整数0-15が許され、論理アドレスの実効サイズをそれぞれ32-17ビットに設定します。アドレスの32ビットすべてが、アドレス変換中に比較されるため、イニシャル・シフトによって無視されるビットがランダム値をもつことはありません。後続のアドレス変換がATCの対応するエントリと一致するようにこれらの値を指定し、また変換テーブル値と一貫性がなければなりません。

テーブル・インデックス(TIA、TIB、TIC、およびTID)

これらの4ビット・フィールドは、変換テーブルで可能な4つのレベルに対するインデックスとして使用する論理アドレスのビット数を指定します(ファンクション・コードでインデックスが付けられたオプション・レベルは含まない)。最高レベルのテーブルに対するインデックスは(ファンクション・コードを使用するときには、それに続いて)TIAで指定され、そして最下位レベルはTIDで指定されます。このフィールドには整数0-15が含まれています。テーブル・サーチ中にTIxフィールド内に“0”を見つけると、ディスクリプタが間接ディスクリプタでなければサーチを終了します。

9.7.3 トランスペアレント変換レジスタ

トランスペアレント変換レジスタ(TT0およびTT1)は32ビット・レジスタで、トランスペアレントに変換される論理アドレス空間のブロックを定義します。トランスペアレントに変換されるブロ

ックの論理アドレスは、修正および保護チェックを行なわず、そのまま物理アドレスとして使用されます。いずれかのTTxレジスタで定義できる最小ブロック・サイズは16Mバイトの論理アドレス空間です。2つのTTxレジスタは、オーバラップするブロックを指定できます。TTxレジスタはTCレジスタのEビットおよび$\overline{\text{MMUDIS}}$信号の状態に関係なく動作します。トランスペアレント変換レジスタを図9-37に示します。

トランスペアレント変換レジスタのフィールドは、次のとおりです。

イネーブル(E)

このビットはこのレジスタで定義するトランスペアレント変換ブロックをイネーブルします。

0－トランスペアレント変換をディセーブル

1－トランスペアレント変換をイネーブル

リセットでこのビットはクリアされます。

キャッシュ禁止(CI)

このビットはトランスペアレント・ブロックのキャッシングを禁止します。

0－キャッシングをイネーブル

1－キャッシングをディセーブル

このビットをセットすると、マッチング・アドレスの内容は内部命令キャッシュまたはデータ・キャッシュに記憶されません。また、このビットがセットされていて、かつマッチング・アドレスがアクセスされると、キャッシュ・インヒビット・アウト信号($\overline{\text{CIOUT}}$)がアサートされ、外部キャッシュに対してこれらのアクセスに対するキャッシングを禁止するよう通知します。

リード/ライト(R/W)

このビットはトランスペアレントに変換されるアクセスのタイプを定義します(マッチング・アドレスに対して)。

0－ライト・アクセスをトランスペアレント

1－リード・アクセスをトランスペアレント

リード/ライト・マスク(RWM)

このビットはR/Wフィールドをマスクします。

0－R/Wフィールドを使用

1－R/Wフィールドを無視

31							24	23			16
論理アドレス・ベース								論理アドレス・マスク			
E	0	0	0	0	CI	R/W	RWM	0	FCベース	0	FCマスク
15							8	7			0

論理アドレス・ベース――トランスペアレント・ブロックを定義するA31-A24の値
論理アドレス・マスク――無視されるビットA31-A24

E　――イネーブル
CI　――キャッシュ禁止
R/$\overline{\text{W}}$　――リード/ライト
RWM　――リード・ライト・マスク
FCベース　――トランスペアレント・ブロックのファンクション・コード値
FCマスク　――無視されるファンクション・コード・ビット

図9-37　トランスペアレント変換レジスタ(TT0およびTT1)のフォーマット

RWMを1にセットすると、マッチング・アドレスのリードおよびライトの両アクセスがトランスペアレントに変換されます。マッチング・アドレスによるリード・モディファイ・ライト・サイクルのトランスペアレント変換の場合は、RWMを1にセットしなければなりません。RWMビットがゼロの場合、どのリード・モディファイ・ライト・サイクルのリードもライトも、TTxレジスタでトランスペアレントに変換されません。

ファンクション・コード・ベース(FC BASE)

この3ビット・フィールドは、このレジスタでトランスペアレントに変換するアクセスのベース・ファンクション・コードを定義します。FC BASEフィールドに一致するファンクション・コードをもつアドレスは、トランスペアレントに変換されます。

ファンクション・コード・マスク(FC MASK)

この3ビットコードにはFC BASEに対するマスクがあります。このフィールドにビットをセットすると、FC BASEフィールドの対応するビットが無視されます。

論理アドレス・ベース

この8ビット・フィールドは、アドレス・ビットA24-A31と比較されます。この比較で一致するアドレス(および適当なアドレス)は、トランスペアレントに変換されます。

論理アドレス・マスク

この8ビット・フィールドには、論理アドレス・ベース・フィールドに対するマスクが含まれています。このフィールドの1ビットをセットすると、論理アドレス・ベース・フィールドの対応するビットが無視されます。16Mバイト以上のメモリ・ブロックは、論理アドレス・マスク・ビットのいくつかを1にセットすることにより、トランスペアレントに変換されます。通常このフィールドの下位ビットをセットして、16Mバイトを超える連続ブロックを定義します。

9. 7. 4 MMUステータス・レジスタ

MMUステータス・レジスタ(MMUSR)は16ビット・レジスタで、PTEST命令の実行により返されたステータス情報を保持します。PTEST命令は、ATC(レベル0のPTEST)または変換テーブル(レベル1-7のPTEST)のいずれかをサーチして、指定された論理アドレスの変換に関するステータス情報を決定します。MMUSRを図9-38に示します。

MMUSRのビットは表9-3に示すように、2種類のPTEST命令に対して異なる意味をもちます。

9. 7. 5 レジスタ・プログラミングの考慮事項

リセット操作が行なわれ、アドレス変換キャッシュ(ATC)のエントリが有効でなくなった場合、ソフトウェアで明示的にフラッシュ(無効にする)操作を指定しなければなりません。$\overline{\text{RESET}}$をアサー

図9-38 MMUステータス・レジスタ(MMUSR)のフォーマット

表9-3　MMUSRの各ビットの定義

MMUSRビット	PTEST, レベル0	PTEST, レベル1～7
バス・エラー(B)	このビットは指定された論理アドレスに対応するATCエントリで、バス・エラー・ビットがセットされている場合にセットされる。	このビットはPTEST命令のテーブル・サーチ中にバス・エラーが発生した場合にセットされる。
リミット(L)	このビットはクリアされる。	このビットはテーブル・サーチ中にインデックスがリミットを超えた場合にセットされる。
スーパバイザ違反(S)	このビットはクリアされる。	このビットは、サーチ中に出会ったロング（S）フォーマット・テーブル・ディスクリプタまたはロング・フォーマット・ページ・ディスクリプタのSビットがセットされており、かつPTEST命令で指定されたファンクション・コードのFC2ビットが1と等しくない場合にセットされる。Iビットがセットされている場合、Sビットは未定義。
ライト・プロテクト(W)	このビットはATCエントリのWPビットがセットされている場合にセットされる。Iビットがセットされている場合は未定義。	このビットはテーブル・サーチ中に出会ったディスクリプタまたはページ・ディスクリプタのWPビットがセットされている場合にセットされる。Iビットがセットされている場合、Wビットは未定義。
無効(I)	このビットは無効な変換を示す。Iビットは、指定された論理アドレスに対する変換がATCにない場合、または対応するATCエントリのBビットがセットされている場合にセットされる。	このビットは無効な変換を示す。Iビットはサーチ中に出会ったテーブルまたはページ・ディスクリプタのDTフィールドが無効になっている場合、またはテーブル・サーチ中にMMUSRのBまたはLビットがセットされている場合にセットされる。
修正(M)	このビットは指定されたアドレスに対応するATCエントリの修正ビットがセットされている場合にセットされる。Iビットがセットされている場合は未定義。	このビットは指定されたアドレスのページ・ディスクリプタの修正ビットがセットされている場合にセットされる。Iビットがセットされている場合は未定義。
透過(T)	このビットは透過変換レジスタ（TT0またはTT1）のいずれか（または両方）で一致があった場合にセットされる。Iビットがセットされている場合は未定義。	このビットはゼロにセットされる。
レベル数(N)	この3ビット・フィールドはゼロにクリアされる。	この3ビット・フィールドにはサーチ中にアクセスされる実際のテーブル番号が入っている。

トすると、TCおよびTTxレジスタのEビットがクリアされ変換がディセーブルされますが、ATCはフラッシュされません。ATCのフラッシュは、新しい値をSRP、CRP、TT0、TT1、またはTCレジスタにロードするPMOVE命令のFDビットの制御のもとでオプションとして行なわれます。

MMUのプログラマは、レジスタをロードすればどのような影響があるか、知っていなければなりません。以下の項では、これらの影響について述べます。MMUSR値は、バス・エラー・ハンドラの適当なルーチンに分岐するためのケース・ストラクチャの使用に適しています。

別の項でこの手法を実現するフローチャートの例を示します。また、9.7.5.3項では、MMU例外を引き起こす条件を説明します。

9.7.5.1　レジスタの二次的効果

PMOVE命令を使用して任意のMMUレジスタ（CRP、SRP、TC、MMUSR、TT0、およびTT1）をロードまたは読み出します。ルート・ポインタ、変換制御レジスタ、またはトランスペアレント・レジスタに新しい値をロードすると、アドレス変換の一部または全部が変化しますので、これらの

レジスタに書込みを行なうときには、ATCの内容をフラッシュしたほうがよいでしょう。CRP、SRP、TC、TT0、およびTT1に書込みを行なうPMOVE命令のオペコードには、これらの命令を実行するときにオプションでATCをフラッシュするフラッシュ・ディセーブル(FD)ビットが含まれています。FDビットが1の場合、命令を実行したときにATCはフラッシュされません。FDビットが0の場合、PMOVE命令の実行中にATCがフラッシュされます。

9. 7. 5. 2　MMUステータス・レジスタのデコーディング

MMUステータス・レジスタ(MMUSR)の7つのステータス・ビットは、それに対してオペレーティング・システムが応答する条件を示します。一般的なバス・エラー・ハンドラ・ルーチンでは、図9-39および図9-40に示すフローを使用して、MMUフォールト(故障)の原因を確認します。PTEST命令がMMUSRのビットを適宜セットすると、プログラムはその条件に対する適当なコード・セグメントに分岐することができます。図9-39にATC(レベル0)に対するPTEST命令のためのフローを示し、図9-40にアドレス変換ツリー(レベル1-7)にアクセスするPTEST命令のフローを示します。

9. 7. 5. 3　MMUコンフィギュレーション例外

MC68030の例外ベクタ・テーブルは、MMUコンフィギュレーション・エラー例外にベクタを割り当てます。コンフィギュレーション例外は、TC、SRP、またはCRPレジスタに無効データをロードしたときに発生します。

Eビットをセットして TC レジスタにある値をロードすると、MMUはすべての4ビット・フィールドにある値の一貫性チェックを実行します。最初の0に出会うまでTIxフィールドの値が加算されます。PSおよびISフィールドの値は、TIxフィールドの合計に加算されます。この合計が32でなければ、PMOVE命令がMMUコンフィギュレーション例外を発生します。この命令は、予約されている値($0-$7)がTCレジスタのPSフィールドに入れられたときにも、コンフィギュレーション例外を発生します。

CRPまたはSRPをロードするPMOVE命令は、DTフィールドの値がゼロ(無効)の場合も、MMUコンフィギュレーション例外を発生します。この場合、例外が発生する前に、レジスタに新しい値がロードされます。

9. 8　MMU命令

MC68030の命令セットには、MMU操作を実行する4つの特権命令が含まれています。これらの命令の詳細については、「第3章　命令セット」を参照してください。

PMOVE命令は、CPUレジスタまたはメモリと6つのMMUレジスタの任意の1つとの間でデータ転送を行ないます。オペレーティング・システムはPMOVE命令を使用し、これらのレジスタを操作したり内容を読み出すことにより、MMU操作の制御と監視を行ないます。オプションにより、PMOVE命令はTC、SRP、CRP、TT0、またはTT1レジスタに何らかの値をロードしたときにATCをフラッシュします。

PFLUSH命令はATCの中のアドレス変換ディスクリプタをフラッシュ(無効にする)します。PFLUSH命令の1バージョンであるPFLUSHAは、すべてエントリをフラッシュします。PFLUSH命令は指定されたファンクション・コードをもつすべてのエントリ、または指定されたファンクション・コードおよび論理アドレスをもつエントリをフラッシュします。

PLOAD命令は指定されたファンクション・コードおよび論理アドレスに対応するテーブル・サーチを実行し、そのアドレスの変換情報をATCにロードします。オペレーティング・システムはこの

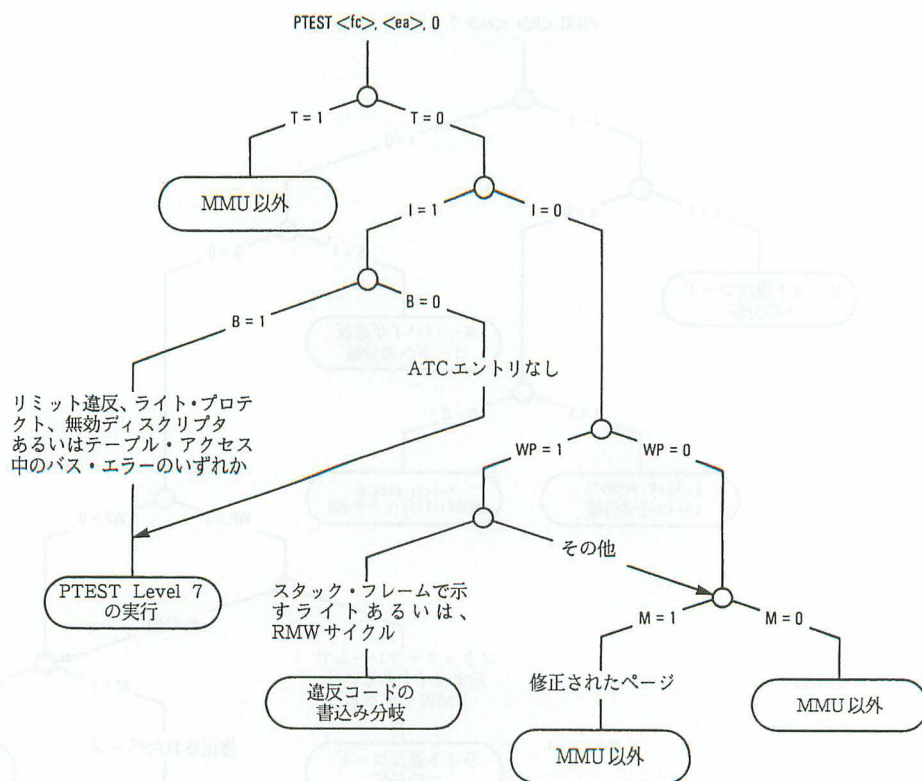



図9-39　PTEST Level 0によるMMUステータスの取得

命令を使用してATCを初期化し、プログラム実行中のテーブル・サーチを最小にすることができます。指定されたアドレスを変換するATCにある既存のエントリは、フラッシュされます。リードまたはライト属性のいずれかに対して、プリロードを実行することができます。ライト属性を選択した場合(PLOADW)、MC68030はテーブル・サーチを実行し、変換テーブルにあるすべてのヒストリ情報(使用および修正ビット)を、あたかもそのアドレスに対するライト操作が行なわれたかのように、アップデートすることができます。同様に、リード属性を選択した場合(PLOADR)、あたかもリード操作が行なわれたかのように、変換テーブルのヒストリ情報をアップデートします。PLOAD命令はMMUSRを変更しません。

PTEST命令はATCをサーチするか、指定されたファンクション・コードおよび論理アドレスのテーブル・サーチを実行し、MMUSRの適当なビットをセットして、サーチ中に出会った状態を示します。最後にフェッチしたディスクリプタの物理アドレスを、アドレス・レジスタに返すことができます。オペレーティング・システムの例外ルーチンは、この命令を使用してMMUフォールトを識別することができます。PTEST命令はATCを変更しません。

この命令は主にバス・エラー処理ルーチンで使用します。たとえば、バス・エラーが発生した場合、ハンドラは次の命令を実行することができます。

PTESTW #1,([A7,offset]),#7,A0

この命令は、MC68030にユーザ・データ空間(#1)のアドレスに対する変換テーブルをサーチし、保護情報をチェックすることを要求します。この論理アドレスは例外スタック・フレーム([A7、offset])から得られます。MC68030はテーブルの最下部(#7――6レベルを超えることはありません)までサーチし、最後に使用されたテーブル・エントリの物理アドレスをレジスタA0に返すよう

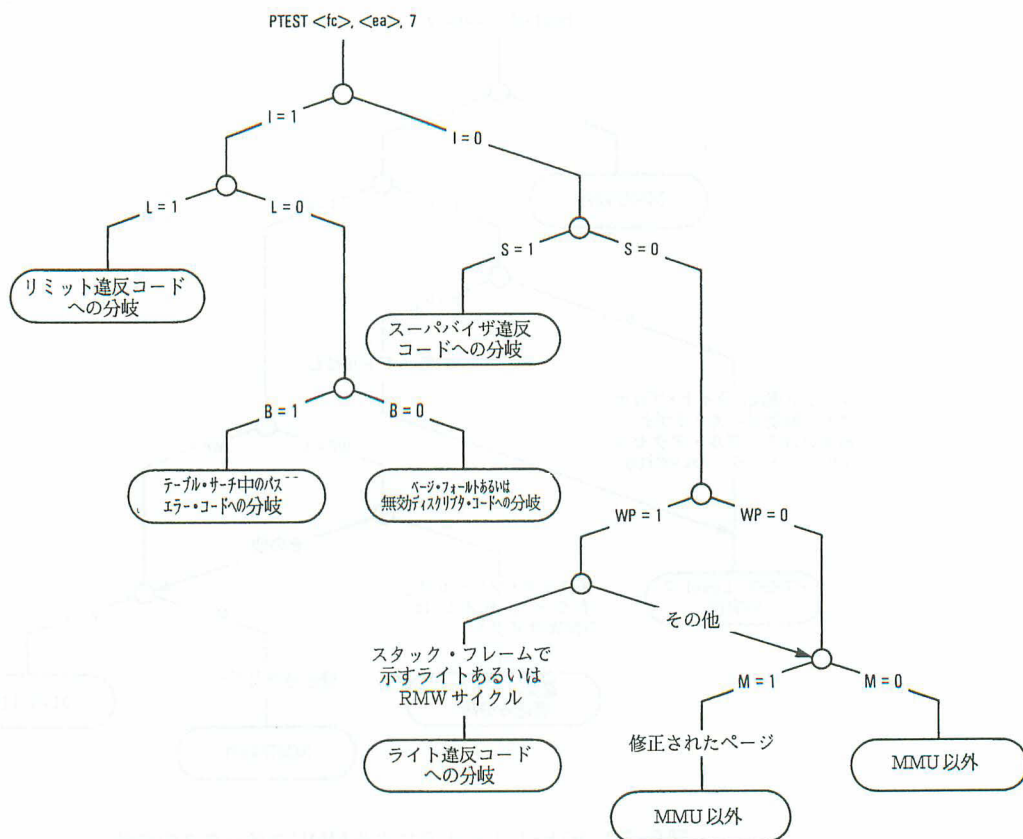



図9-40　PTEST Level 7によるMMUステータスの取得

指示されます。この命令を実行した後、ハンドラはMMUSRを調べてフォールトの原因を見つけ、A0を使用して最後のディスクリプタにアクセスすることができます。なお、PTESTRおよびPTESTW命令は、TTxレジスタが論理アドレスに一致するか、そのレジスタのR/Wビットがマスクされていなければ、PTEST0を除いて同じ結果になります。

MMU命令は対応するMC68851の命令と同じオペコードとコプロセッサidを使用します。MC68030がサポートしないCP-ID＝0をもつすべてのFライン命令(MC68851の命令を含む)は、スーパバイザ・モードで実行しようとすると、Fライン未実装命令例外を発生します。CPID＝0の未実装Fライン命令をユーザ・モードで実行しようとすると、MC68030は特権違反例外を発生します。ゼロ以外のコプロセッサ idをもつFライン命令は、コプロセッサ命令と同じようにMC68030で実行されます。

9.9　オペレーティング・システムでのページ・テーブルの定義と使用法

オペレーティング・システムでMMUをどのように使用するかを決めるには、多くの要素を考慮しなければなりません。MC68030はさまざまなシステム・インプリメンテーションに対してオペレーティング・システムを最適化できるようにするために必要な柔軟性を備えています。次項で説明するオペレーティング・システム例は、オペレーティング・システムの設計への1つのアプローチであり、多くのトレード・オフについて検討しています。

9.9.1　ルート・ポインタ・レジスタ

オペレーティング・システムは、CPUルート・ポインタ(CRP)レジスタだけ、あるいはCRPレジスタとスーパバイザ・ルート・ポインタ(SRP)レジスタの両方を使用して、最上位アドレス変換テーブルを指すことができます。この選択は、システムのメモリ・レイアウトの複雑さによります。CRPだけを使用するときには、CRPはすべてのスーパバイザおよびユーザ・リファレンスをマップする変換テーブルを指し示す必要があります。しかしながら、CRPレジスタしか使用しないときでも、スーパバイザ・テーブルとユーザ変換テーブルを分離することができます。最上位レベルの変換テーブルへのインデックスがファンクション・コード値のとき(TCレジスタのFCLがセット)には、スーパバイザおよびユーザ・テーブルはすべての下位レベルで分離されます。アドレス・テーブルを適切に正しく構成することにより、どちらの方法も同じ機能性を実現できますが、それぞれに利点があります。

変換テーブルがCRPおよびファンクション・コード・ルックアップを使用するときには、スーパバイザおよびユーザ・アクセスは分離され、各タスクは別々のスーパバイザおよびユーザ・マッピングをもつことができます。

また、各タスクのスーパバイザ空間に対応するファンクション・コード・テーブルのエントリは、すべて同じテーブルを指し示すことができ、すべてのスーパバイザ参照に対して共通のマッピングを与えます。

スーパバイザ・アドレス空間のマッピングがすべてのタスクで同じとき、CRPとともにSRPを使用することにより、マッピングを定義するためのより簡単で効率のよい方法を提供することができます。この手法はファンクション・コードの使用を抑止し(プログラムおよびデータ空間が明白なマッピングを要求しないかぎり)、変換テーブルのルート・ポインタ・レベルでのスーパバイザおよびユーザ・アクセスを分離します。単一の変換テーブルは、各タスクに対する変換テーブルに多数のスーパバイザ・ポインタをもつことなく、すべてのスーパバイザ・アクセスをマップするため、テーブル・サーチのためのバス動作が低減されます。

9.9.2　タスク・メモリ・マップの定義

MC68030は、スーパバイザがユーザ・アドレス空間をアクセスするための手段をいくつか提供します。スーパバイザは、仮想空間がどのように分割されているかに関係なく、MOVES(空間の転送)命令を使用して、どのユーザ・アドレスにでもアクセスすることができます。システムによっては、各タスクに対して完全な4ギガバイトの仮想記憶マップを備えているものもあります。実際、仮想マシン環境で他のオペレーティング・システムを動作させるオペレーティング・システムは、従属オペレーティング・システムの完全なアドレッシング範囲を正確にエミュレートする完全なマップを備えなければなりません。

MC68030の大きなアドレス空間を用いて、個々のユーザ・タスクあるいはすべてのユーザ・タスクがオペレーティング・システムとアドレス空間を共有することができます。この機能を実行する1つの方法は、次項のオペレーティング・システム例にインプリメントされています。アドレス空間を共有すると、オペレーティング・システムでユーザ・データに直接アクセスすることができます。このマッピング方式のもう1つの利点は、タスクが容易にコードを共有できることです。ファイルI/Oハンドラなどの共通ルーチンおよび算術変換パッケージをリエントラント形式で書き、システムのすべてのユーザ・タスクからのアクセスを読出し専用アクセスに制限することができます。

共用仮想アドレス空間システムの最も簡単な例は、各ユーザおよびスーパバイザ・プロセスに、1つの4ギガバイトの仮想アドレス空間内で、固有の仮想アドレス範囲が与えられているシステムです。言い換えれば、このシステムは1つのリニア仮想アドレス空間だけをもち、すべてのプロセスは

空間内のどこかで動作します。全体のシステムに対して変換テーブル・ツリーが1つだけ必要ですが、各タスクは必要に応じて個々のテーブルをもつことができます。共通ツリー・アプローチにより、オペレーティング・システムはルート・ポインタを修正することなく、どのタスクのどのアイテムでもアクセスできます。そうでない場合は、現在アクティブになっているタスクだけがすぐにアクセス可能で、通常はこれで十分です。タスクを切り換えるには、オペレーティング・システムは、ファンクション・コードでインデックスが付けられた最上位レベルの変換テーブルにあるユーザ・プログラムおよびユーザ・データ・ポインタだけをアップデートしさえすればよいのです。この方式はテーブル管理が容易であり、アイテムにシステムのすべてのタスクに対する同じ仮想空間を与えることにより、たやすく共通アイテムを共有することができる利点があります。この方式はメモリ管理機構に、それ以上の複雑さを求めないリアルタイム・システムに最適です。

オペレーティング・システムをさらに複雑にし、ユーザおよびスーパバイザ仮想メモリ・マップを共有するために、次の論理ステップは、スーパバイザ・アドレスを分離したままにするが、各ユーザ・タスクに残りの仮想空間をおのおのに割り当てることができます。たとえば、各ユーザ・タスクは0から512メガバイトの仮想記憶空間をもつことができ、オペレーティング・システムのプログラムおよびデータは512メガバイトから最大4ギガバイトのサイズの残りの空間を占有するようなことがあるわけです。各ユーザ・タスクは独自の変換テーブル群をもっています。スーパバイザ・ルート・ポインタは、ユーザ・テーブルがスーパバイザ空間もマップするか否かにより、使用する場合もあり使用しないこともあります。

前述の方法のように、ユーザはオペレーティング・システムが許可するか、共通ルーチンを共有しようとしないかぎり、オペレーティング・システム部分にアクセスすることはできません。この方式の利点は、各ユーザ・タスクに対してより大きな仮想アドレス空間を提供し、そして仮想メモリの断片化の問題を回避することにあります。この方式の欠点は、若干複雑なテーブル管理を必要とし、またオペレーティング・システムのアクセスが現在のユーザ・タスクに限定されることです。

MC68030MMUを使用する上で必ず守らなければならないルールがいくつかあります。一般に、前述した方式を用いてオペレーティング・システムのアクセスを1つのユーザ・タスクに限定することは、それに該当します。ただし、4ギガバイトの仮想アドレス空間全体を用い、アドレス空間をクロス・マッピングすることにより、スーパバイザは独自のスーパバイザ・マップの特定部分として、各ユーザ・タスク空間にアクセスすることができます。各ユーザ・タスクが16メガバイトの仮想アドレス空間に制限され、スーパバイザが16メガバイトのアドレス空間しか必要としない場合は、256のそのようなアドレス空間を同時にマップ可能です。スーパバイザ変換テーブルには、これらの各空間を含めることができ、スーパバイザは、特定のタスクに対する正しい定数を保持するレジスタを用いたインデックス・アドレッシングを使用して各タスクにアクセスすることができます。この定数はスーパバイザ-ユーザ仮想アドレス変換を与えます。システム・プログラマは、MMUの柔軟性を引き出すきわめて高度な機能をいくつかインプリメントすることができます。

最も複雑なシステムおよび仮想マシン能力をインプリメントするシステムは、スーパバイザおよびすべてのユーザ・タスクの仮想アドレス空間を完全に分離するか、あるいは可能であれば個別のスーパバイザ・タスクの仮想アドレス空間をも完全に分離することができます。各ユーザまたはスーパバイザ・タスクは、ゼロからスタートし4ギガバイトまで広がる、独自の仮想メモリ空間をもっています。ファンクション・コードを使用して各タスクに対し、プログラムに4ギガバイトのアドレス空間を与え、データに別の空間を用意することができます。各種空間の間には共通するものがないため、SRPおよびCRPの両方を使用することになるはずです。オペレーティング・システムは、MOVES命令を使用してユーザ空間とのやりとりを行ないます。この方法の利点は仮想空間を最大限に活用し、アドレスを完全に論理的に分離できることです。仮想マシンをインプリメントするには、仮想空間を最大限に活用することが必要です。欠点としては、テーブル管理が複雑になり、他

のアドレス空間へのアクセスが制限されることです。

9.9.3　テーブルの定義に対するMMU機能のインパクト

通常、タスクに対するメモリのマップ方法を決定した後、テーブルの定義にインパクトを与えるMMUの機能を考慮することになります。システムによっては、これらの機能がマッピングの決定に影響を与えるため、マッピングを決めるときに考慮しなければなりません。

9.9.3.1　テーブル・レベル数

MMUはアドレス変換テーブルでゼロから5レベル(インダイレクションを使用して6レベル)をサポートします。ゼロ・レベルは、ルート・ポインタにおけるアーリ・ターミネーションの場合です。これはシステムに対する物理アドレス範囲にリミット・チェックを与えます。主に物理アドレスに対してリミット・チェックが必要なシステムで用います。

大きなページ・サイズをサポートするシステム、または限定された仮想メモリ空間しか必要としないシステムは、単一レベルのテーブルを使用することができます。32Kバイトのページをもつ単一レベルの変換ツリーは、基本的に仮想ページ・フォールトやページングI/Oのオーバヘッドを最小限にする必要のある、数値計算を主体としたシステム(すなわち、データの移動ではなく演算操作を実行するシステム)に対して最善の選択です。このタイプのシステムは、わずか2Kバイトのページ・テーブル空間で16メガバイトのアドレス空間をマップすることができます。マップを多用したアドレス空間により、テーブル・サーチ時間が低減されます。

もう1つの極端な例としては、2メガバイトの仮想アドレス空間しか必要としないシングル・ユーザのビジネス・システムがあげられます。多くのウィンチェスタ型ハード・ディスク・ファイル・システムのブロック・サイズ・フォーマットが512バイトになっているため、このシステムには512バイトのページ・サイズが最適と考えられます。2メガバイト空間を完全にマップするページ・テーブルは、わずか16Kバイトのメモリしか必要とせず、ATCエントリは一度に11Kバイトの仮想空間を直接マップします。このシステムのページ・テーブル、および前項で説明したシステムはサイズが小さいため、オペレーティング・システムのデータ領域の中に固定的に割り当てることができます。これらは実質的に管理またはスワッピング・オーバヘッドは生じません。

2レベルのアドレス変換テーブルは、前項で述べたページ・テーブルに類似した下位ページ・レベルと上位レベルでの追加ディレクションを提供します。たとえば、32Kバイトのページと512エントリのページ・テーブルを使用するシステムでは、上位レベルの変換テーブルには256エントリのショート・フォーマット・ディスクリプタがあり、テーブルに1Kバイトを必要とします。上位テーブルの各エントリは、仮想アドレス空間の16メガバイト領域をマップします。大規模な“数値主体型”システムに対する2レベル・テーブルの主な利点は、オペレーティング・システムの設計者がページ・サイズとテーブル・サイズ間にトレードオフをもたせることができることです。システム設計者は、使用可能なI/Oデバイスにブロック・サイズをフィットさせるために、小さなページ・サイズを選択しながら、テーブルを管理することができます。ただし、設計者はページ・サイズを小さくしたために生じる性能上のペナルティに留意する必要があります。ページ・サイズが小さいシステムでは、ページ・フォールトの頻度が高いため、テーブル・サーチおよびI/Oのページングに時間がかかります。MC68030は柔軟性が高いため、設計者にはテーブル構造の設計およびページ・サイズを最適化するための十分な選択が与えられています。

3レベルの変換テーブルは、オペレーティング・システムが共有メモリ空間や共有ページ・テーブルを頻繁に使用するときに有用です。高度なシステムは、しばしば変換テーブルまたはプログラム、およびページ・テーブル・レベルで定義されたデータ領域を共有します。テーブル・エントリが、別のタスクでも使用する変換テーブルを指し示すことができるときには、メモリ領域の共有が効率的

になります。スーパバイザによるユーザ・アドレス空間への直接アクセスはメモリ共有の一例です。

人工知能型のシステムには、このように広範な広がりをもつアドレス間に、ごくわずかなメモリだけを割り当てた超大規模の仮想アドレス空間を必要とするものもあります。このメモリの断片化は、システムがデータ・リストに対して実行する複雑な再帰的な動作によるものです。これらの動作はシステムに、メモリ・マップ内で複雑なポインタおよびリンク・リストを絶えず割り当てたり、解放したりすることを要求します。この断片化は、メモリを最も効率よく利用するために、小さなページ・サイズを提案します。しかし、大規模な仮想メモリ・マップ内の小さなページ群は、比較的大きな変換テーブルを必要とします。たとえば、256バイトのページをもつ4ギガバイトの仮想アドレス空間をマップするには、ページ・テーブルだけで64メガバイトを必要とします。3ないし4レベルのテーブル構成では、実際の変換テーブル・エントリ数を大幅に低減することができます。設計者は無効ディスクリプタを使用して、未使用アドレスのブロックを表わし、有効ディスクリプタのリミット・フィールドを使用してポインタとページ・テーブルのサイズを最小限にすることができます。さらに、アドレス・テーブル自体のページングにより必要なメモリを低減します。

9.9.3.2 イニシャル・シフト・カウント

変換制御レジスタ(TC)のイニシャル・シフト・フィールド(IS)は、変換テーブルのサイズを減らすことができます。必要な仮想アドレス空間が32ビット以下でアドレス指定できるときには、ISフィールドは、最上位論理アドレス・ビットの適当数を廃棄することによって、仮想アドレス空間のサイズを低減します。この手法は非常に大きな違法(つまり、領域外)アドレスを検出するシステムの能力を抑止します。全32ビット・アドレスを使用し、無効ディスクリプタおよび制限付きポインタ、そしてページ・テーブル・サイズによりテーブル・サイズを減らすと、この問題を防止することができます。

9.9.3.3 リミット・フィールド

ファンクション・コードでインデックスを付けたテーブルを除き、各ポインタおよびページ・テーブルは定義されたサイズをもつことができます。リミットを定義するとオペレーティング・システムの柔軟性が向上し、変換テーブルのメモリを節減します。テーブル・ディスクリプタのリミット・フィールドは、指し示すテーブルのサイズを制限します。このリミットは上限または下限のいずれかで、テーブルの範囲内で下位または上位アドレスのどちらかを使用します。タスクが可能な最大数の仮想ページを必要とすることはほとんどないため、この程度のテーブル・サイズの低減は実用的です。

たとえば、オペレーティング・システムが4Kバイトのページを使用し、それぞれが平均で80Kほどのサイズの小さなタスクを多数動作させるときには、各タスクは20エントリのページ・テーブルを1個必要とします。システムは各テーブルのサイズを80バイト、つまり10のタスクに対して800バイトに制限することができます。制限しなかった場合、これら10のタスクを動作させるオペレーティング・システムは、ページ・テーブルだけでも40Kバイトの空間(1ページに1テーブル)を必要とすることになります。

変換テーブルに求められるメモリの節減は、きわめて大きなメモリ・マップを必要とする傾向にある人工知能システムに対しては特に重要です。リミット・フィールドを使用することにより、各テーブルはその中にあるアクティブ・エントリ数と同じ大きさになります。この制限はテーブルが大きくなるに従って変更できます。上位レベルのテーブルでは、各テーブルは追加エントリが必要なときにしか大きくなりません。3ないし4レベルのテーブルを使用すると、これらのテーブルの管理が容易になります。

9.9.3.4 アーリ・ターミネーション・ページ・ディスクリプタ

ポインタ・テーブルにあるページ・ディスクリプタは、ページ全体のブロックをマッピングするアーリ・ターミネーション・ページ・ディスクリプタです。つまり、仮想アドレスの連続範囲を物理アドレスの連続範囲にマップします。たとえば、オペレーティング・システムは、特別なスーパバイザI/O周辺デバイスに32Kバイトの領域を確保することができるわけです。この領域を単一のアーリ・ターミネーション・ディスクリプタでマップして、変換テーブル・サイズおよびテーブル・サーチ・オーバヘッドを節減することができます。ディスクリプタは、ブロック・サイズが特定のディスクリプタが表わす仮想アドレス空間よりも小さいときには、リミット・フィールドを使用して連続ブロックのサイズを低減することができます。MC68030は、ページがアクセスされるときに、アーリ・ターミネーション・ディスクリプタで表わされる仮想アドレス範囲に対して、複数のATCエントリ(各ページに1つ)を生成します。

オペレーティング・システムは、アーリ・ターミネーション・ページ・ディスクリプタを使用して、各タスク(プログラムおよびデータ)に対する連続メモリ・ブロックをマップすることができます。このタスクは、ディスクリプタの物理アドレス部分を変更することにより、リロケートできます。この方式はシステムのタスクがグループとしてスワップ可能な1つまたはいくつかのシーケンシャル・メモリ・ブロックで構成されているときに有用です。オペレーティング・システムのメモリ・マップは、これらのブロック内のアドレス空間全体を、すべてのタスクに利用可能な均一仮想空間として扱うことができます。このシステムは1つの変換テーブルしか必要とせず、リミット・フィールドとアーリ・ターミネーション・ページ・ディスクリプタを使用して、完全なメモリ・セグメントをマップします。

9.9.3.5 間接ディスクリプタ

間接ディスクリプタはページ・テーブルにあるテーブル・ディスクリプタです。間接ディスクリプタは、変換ツリーにある別のページ・ディスクリプタを指し示します。ページに対応する間接ディスクリプタを使用して、そのページをいくつかのタスクに共通なものにすることができます。共通ページのヒストリ情報は1つのディスクリプタでのみ維持されます。このページにアクセスすると、使用(U)ビットがセットされ、そのページにライト操作を行なうと、ページのM(修正)ビットがセットされます。オペレーティング・システムが利用可能なページをさがしてサーチを行なうときは、共通ページに対するディスクリプタをもつページ・テーブルをチェックするだけで、そのステータスを確認します。

他のページ共有方法では、システムはすべてのタスクに対するページ・テーブルをチェックして、共用ページのステータスを決定しなければなりません。

9.9.3.6 未使用ディスクリプタ・ビットの使用

一般に、多数のタイプのディスクリプタの未使用フィールドのビットは、オペレーティング・システムが独自の目的に使用することができます。特に、無効ディスクリプタは、そのフォーマットで利用できる32ビット(ショート)または64ビット(ロング)のうちの2ビットだけを使用します。オペレーティング・システムは、通常これらのフィールドを、仮想アドレス空間が割り当てられているかどうか、そしてイメージがページング・デバイスに存在するかどうかを示すソフトウェア・フラグとして使用します。これらのフィールドには、イメージの物理アドレスが含まれている場合もよくあります。

また、オペレーティング・システムは、メモリ内のページに関する情報を未使用フィールド中で維持することがよくあります。この情報はエージング・カウンタまたはページの使用頻度を表示す

るものです。この情報はページを再割当てした場合に、オペレーティング・システムが、システム性能に最も影響の少ないページを識別するのを手助けします。このシステムは最初に、仮想ページに割り当てられていない物理ページ・フレームを使用しなければなりません。次に、最後にアクセスされてから最長経過時間をもつページを使用します。M(修正)ビットがセットされていないページを最初に取り出します。それは、そのページをページング・デバイスにコピーする必要がないためです(既存のイメージは有効になったままです)。

ページ・ディスクリプタの未使用フィールドの中で、エージング・カウンタをセット・アップすることができます。システムはページに対するU(使用)ビットを周期的にチェックして、そのページが前回のチェック以来使用されてないときに、カウントをインクリメントします。このシステムは、エージング・カウンタのカウント値から最も以前に使用されたページを識別することができます。あるページのカウンタがオーバフローすると、システムは最も以前に使用されたページのキューにあるページをリストし、それから再割当てする次のページを選択することができます。

オペレーティング・システムの設計者に対して、取り出すページを選択するためのさまざまな方式が提供されています。あるオペレーティング・システムは、最も優先順位の低いタスクからページ・テーブルを操作し、スチールする古いページをさがします。別のシステムは、すべてのページ・フレームの使用に応じたシステム規模のリストを維持し、それらを使用して最も古いページからリストの操作を始め、スチールするページをさがします。高度なシステムは各個別タスクに対するアクティブ・ページのワーキング・セット・モデルを保持します。システムはこの情報から、1回のI/O操作で完全なページ・ブロックをスワップ・インおよびスワップ・アウトすることができます。選択した方法は、使用頻度の高いシステムにおいて、ページ・フォールト・オーバヘッドを制限する上で重要な影響をもつことができます。

9. 10 オペレーティング・システムへのページングのインプリメンテーション例

この章では、MMUの機能のいくつかを示すオペレーティング・システムの設計例を述べます。この説明は設計のバリエーションを提供する別の方法を提案します。実際のコードを引き出すためにインプリメント可能なメモリ管理アルゴリズムを示しています。また、バス・エラー・ハンドラ・ルーチンも示しています。アルゴリズムのインプリメントにより、オペレーティング・システムのメモリ管理サービスに対する基本コードを開発しています。

9. 10. 1 システムの説明

例としてとりあげたシステムは、大規模な仮想メモリ・タスク空間をマップする能力をもっており、これは数値主体型の処理タスクの実行に必要です。これらのタスクの大部分は、16メガバイト以上のメモリを必要としませんが、システムはより多くのメモリを必要とする一時的なタスクに対して大きな仮想メモリ空間(496メガバイトまで)を供給することができます。このシステムはトラッシングおよび変換テーブル・サーチを少なくするために、8Kバイトという比較的大きなページ・サイズを使用します。大きなページ・サイズの場合、少ないディスクリプタで仮想メモリの広い領域をマップすることができます。また、どの時点でもMC68030にとっては、ATCミスの発生が少なく、そしてテーブル・サーチの実行も少なくてすみます。

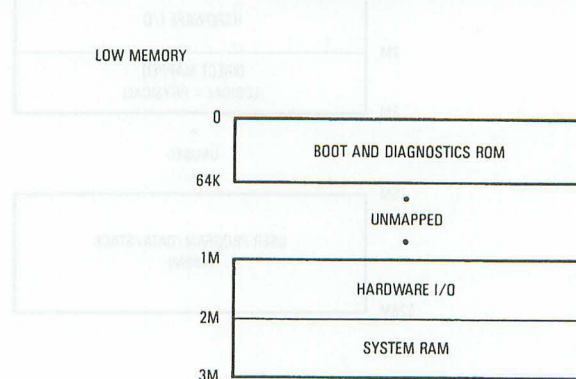
ページ・サイズが大きくなると、大きなデータ・ブロックを転送するためのページングI/O操作を必要とし、場合によってはページのごく一部しか実際に使用しないこともあります。しかし、予備ソフトウェア・モデル・シミュレーションは、8Kバイトのページがこの種の処理に対して最適な性能を与えることを示しています。

このシステムに対する平均タスクは、192Kバイトのメモリ、つまり24個の8Kバイト・ページを必要とするコンパイラ、またはテキスト・エディタです。ショート・ページ・ディスクリプタを使用すれば、このページ・テーブルは96バイトを占有します。

ページ・テーブルは任意の16バイト境界に常駐することができます。MMUのリミット・フィールドは余分な空間を必要とせずに、所要領域を供給することができます。そのためアドレス・テーブル領域を小さくして、完全に物理メモリに常駐させることができます。オペレーティング・システムはテーブル領域をページ化する必要はありません。

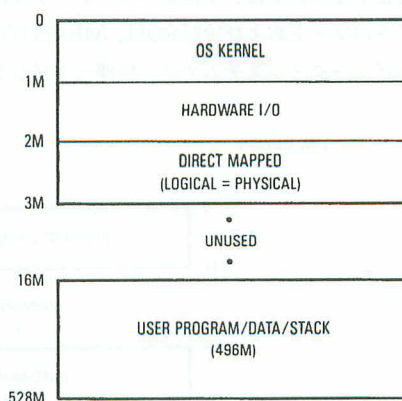
多数のコンピュータ・システムのページング・ハードウェアは、下位レベルのテーブルをページ境界に常駐することを必要とし、効果的に1つまたは複数のページ全体を活用します。これは、10個のタスク(1テーブル当たり、1個の8Kバイト・ページ)に対するページ・テーブルに80Kバイトを必要とします。そして、上位レベルのテーブルに必要なメモリが、1タスク当たり最小8Kバイトで追加されると、合計は160Kバイト以上になります。MC68030のテーブル・ベース・アドレスはゼロ・モジュロ16アドレスです。その結果、アドレス・テーブル空間のメモリを大幅に節減することができます。10個のタスクのためのページ・テーブルに対して80Kバイトを使用する代わりに(10個のテーブルに、1テーブル当たり8Kバイトのページを1個ずつ)、MC68030は960バイトを必要とします。上位レベルのテーブルについて、1タスク当たり8Kバイトを与える代わりに、MC68030ではテーブルは2560バイトを必要とします。小さいテーブルを割り当てるときに発生するおそれのある断片化によって、所要メモリ量が増大する可能性がありますが、それでも160Kバイト以下に抑えることができます。

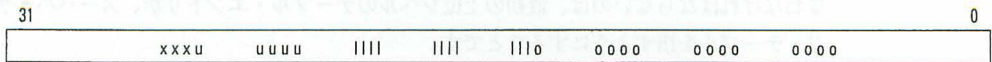
例にとりあげたシステムの変換テーブル・ツリーは、2つのレベルからなります。上位レベルは32エントリを含む固定テーブルで、各エントリは下位レベルのページ・テーブルを指し示すロング・フォーマットのテーブル・ディスクリプタです。各ページ・テーブルは16メガバイトまでの仮想アドレス空間をマップします。上位レベル・テーブルは小さい(256バイト)ため、タスクの主制御ブロックに容易に適合します。このシステムが新しいタスクをデスパッチすると、そのタスクに対する上位レベル・テーブルへのポインタをCRPレジスタにロードします。各下位レベルのテーブルは、0から2048のショート・フォーマット・ページ・ディスクリプタで構成されます。ページ・テーブルに対するテーブル・ディスクリプタのリミット・エントリが、テーブルのサイズを決定します。平均192Kバイトのタスクに対しては、上位レベルのテーブルは通常1つの有効エントリをもち、そしてこのエントリは96バイトの平均サイズをもつ下位レベルのテーブルを指し示します。16メガバイト以上を必要とするタスクは、上位レベルのテーブルに2つ以上の有効エントリを必要とします。

64Kバイトのブートおよび診断ROM、64KのI/O領域、および1メガバイトのRAMを備えた標準的なコンピュータ・システムでは、物理マッピングは次のとおり現われます。

オペレーティング・システムは仮想メモリのページを保持するために、物理メモリ(ページ・フレーム)に対するメモリの割当てを制御しなければなりません。すべての利用可能な物理メモリがページ・フレームに分割され、それぞれが仮想メモリのページを保持することができます。4メガバイトの実メモリをもつシステムは、512の8Kバイト・フレームに分割され、理論上一度に512ページのアクティブな仮想メモリを保持することができます。通常、オペレーティング・システムのコンポーネント(例外ハンドラ、カーネル、プライベート・メモリ・プール)は、恒久的にメモリの一部に常駐します。残りのページ・フレームしか仮想メモリ・ページに使用できません。

オペレーティング・システムはすべての未割当てのページ・フレームのリンク・リストを維持します。これを行なうための簡単な方法の1つは、各未割当てのフレームに次のフレームへのポインタをもたせることです。オペレーティング・システムはフレームが要求されると、リストの最初のページ・フレームを取り出します。GetFrameとよぶオペレーティング・システム・プリミティブはこの機能を実行し、使用可能なフレームの物理アドレスを返します。全フレームが割り当てられると、GetFrameは別のタスクからフレームをスチールします。GetFrameは最初に未修正のフレームをさがしてスチールします。未修正のフレームは、仮想ページ・イメージを格納する外部記憶デバイスに、そのページをコピーバックするのを待たずにスチールされます(このデバイスをページング・デバイスまたはバッキング・ストアとよびます)。未修正のページ・フレームがない場合、GetFrameはシステムが修正済みのページをページング・デバイスにコピーする間待ち、それからページ・フレームをスチールして、呼出し側に物理アドレスを返します。

次に、オペレーティング・システムは、スーパバイザ・ワーク・メモリを割り当てたり、解放するための物理メモリ管理ルーチンを必要とします。このルーチンは少なくともアドレス変換テーブルに必要な、モジュロ16の境界にメモリ部分を割り当てなければなりません。通常、このタイプのルーチンは特定のサイズのメモリ部分を割り当てます。GetRealは割当てルーチンで、ReturnRealはリターン・ルーチンです。これらのルーチンは物理アドレスを使用します。

用意された物理メモリ割当てにより、オペレーティング・システムはすべてのタスクに対する仮想メモリを管理できなければなりません。これを行なうために、システムは仮想メモリ・マップを知らなければなりません。つまり、仮想メモリ空間の合計量、割当て量、およびタスクに割当て可能な領域について知らなければなりません。仮想メモリ・マップは次のような構造になっています。

この仮想メモリに対する仮想アドレスは次のように細分されます。

```
31                                                                    0
+---------------------------------------------------------------------+
|           xxxu   uuuu   llll   llll   lllo   0000   0000   0000     |
+---------------------------------------------------------------------+
```

x ――無視(3ビット)

u ――上位レベル・インデックス(5ビット)、32のロング・テーブル・エントリをマップ

l ――下位レベル・インデックス(11ビット)、2048のショート・ページ・エントリをマップ

変換テーブル構造は次のもので構成されます。

CRP →タスク制御ブロックの上位レベル・テーブルで、32のロング・ポインタを含む。

[0] →すべてのタスクに共通な下位レベル・テーブルで、すべてのオペレーティング・システム領域(仮想空間の最初の4メガバイト)をマップする。この共通テーブルは512のショート・ページ・エントリ(2Kバイト)を含む。

[1] →ユーザ・プログラム/データ/スタック領域の最初の16メガバイトに対する下位レベル・テーブル。

⋮

[31] →ユーザ・プログラム/データ/スタック領域の最後の16メガバイト(合計496)に対する下位レベル・テーブル。

ユーザ・プログラムは、仮想アドレスを16メガバイトから開始して512メガバイト限界まで上位方向にのみアクセス可能です。ユーザ・タスクのコード、データ、およびスタックは仮想メモリのこの領域に割り当てられます。スーパバイザ・プログラムは仮想マップ全体にアクセスできます。これらは、直接I/Oポートにアクセスするアドレスおよび未変換アドレスの物理メモリ全体にアクセス可能です。アドレス・テーブルは、仮想アドレスが1～3メガバイトの間のスーパバイザ用物理アドレスと等しくなるようにセットアップされます。物理アドレス空間を仮想空間にたたみ込むと、物理アドレスを使用する操作が大幅に簡略化されます。このたたみ込みは必ずしも仮想アドレスが物理アドレスと同じであることを意味しません。たとえば、物理アドレス・ゼロにあるブート/診断ROMには3メガバイトの仮想アドレスを割り当てることができます。しかし、バスの物理サイドにある外部バス・マスタや外部回路(ブレークポイント・レジスタなど)は物理アドレスをもっていなければなりません。これはオペレーティング・システム・コードがアドレス変換を実行するためにオーバヘッドを必要とします。

この仮想メモリ・マップはユーザ・アドレスとは異なるスーパバイザ・アドレスを備えています。すべてのスーパバイザ・ルーチンは、特定の命令やアドレッシング・モードに制約されることなく、任意のユーザ領域にでも直接アクセス可能です。ユーザ・マップとスーパバイザ・マップが別々になっているため、スーパバイザ・マップ用とユーザ・マップ用に1つずつ計2つのルート・ポインタを使用することになります。しかし、スーパバイザはユーザ・データ・アイテムに正しくアクセスするために、ユーザ変換テーブルにアクセスできなければなりません。別々のルート・ポインタにより、スーパバイザ・テーブル構造をユーザのテーブル構造に結合しなければなりません。これを行なうには、スーパバイザ・アドレス・テーブルに対するテーブル・ルックアップの追加レベル(ファンクション・コード・レベル)が必要です。

この例では、代わりに簡単な方式を採用しています。CPUルート・ポインタだけを使用し、各タスクに対して、上位レベル・テーブルの最初のエントリ(スーパバイザ部分に対する、仮想アドレス空間の最初の16メガバイト)は同じ下位レベル・テーブルを指します。この共通の下位レベル・テーブルはスーパバイザ保護をもち、仮想オペレーティング・システム全体、物理I/O、および物理メモリ領域をマップします。この方式はタスクの切換え時にユーザ/スーパバイザ境界をまたいで正しいアクセスを行なうために、余分なルックアップ・レベルやポインタ操作の必要をなくしています。

新しいタスクのアドレス・テーブルを生成するときに、すべてのオペレーティング・システムが行なわなければならないのは、最初の上位レベルのテーブル・エントリが、スーパバイザの共通ページ・テーブルを指すようにすることです。

オペレーティング・システムは、ユーザ・タスクに割り当てられている仮想メモリ領域に対する説明上の問題を解決するために、既存の変換テーブルを使用してこれらの領域を識別します。有効なディスクリプタが任意の仮想アドレス・ページを指すときは、8Kバイト・ページのメモリが割り当てられています。この方式は8Kバイトのページ・サイズの倍数のメモリ領域を与えます。8Kの細分性のために、この方式は絶えず仮想メモリ空間の要求と返却を行なうタスクにとっては不十分です。そのため、別の手法を使用することになります(おそらく、仮想空間の可用性を示す補助テーブル)。このシステムのタスクがメモリ空間の追加を要求することはほとんどありません。出される要求はすべて大きな領域に対するものです。この方式はこれで十分です。UNIX(r)の環境で動作するアプリケーション・プログラムおよびユーティリティもメモリに対して同様な要求をもっています。

オペレーティング・システム・プリミティブ GetVirtualは、タスクに仮想メモリ空間を割り当てます。入力パラメータはバイト単位のブロック・サイズであり、新しいブロックに対して仮想アドレスを返します。GetVirtualは最初に、要求されたサイズが大きすぎないかどうかチェックします。次に、変換テーブルを操作して要求されたブロックを保持できるだけの大きさの未割当て仮想メモリ領域をさがします。十分な空間が見つからなかった場合、GetVirtualはページ・テーブル・サイズをその最大値まで増やします。それでも空間を与えられなければ、GetVirtualはエラー表示を返します。このルーチンがブロックに対して十分な仮想空間を見つけたときは、そのブロックのページ・ディスクリプタを未使用の状態(無効であるが、割当て済み)にセットします。これらのページを最初に使用すると、ページ・フォールトが発生します。オペレーティング・システムは、そのページにページ・フレームを割り当て、ディスクリプタを有効なページ・ディスクリプタと置き換えます。ステータス(無効ディスクリプタのソフトウェア・フラグで示す)は、オペレーティング・システムに対し、ページング・デバイスはこのページに対するページ・イメージをもたないことを知らせます。したがって、このページング・デバイスからのリード操作は不要です。

無効ディスクリプタのステータスが、ページ・イメージを読み込まなければならないことを示しているときには、プリミティブ SwapInPage がそのイメージを読み込みます。このルーチンの入力パラメータは無効ディスクリプタで、その中にはページ・イメージのディスク・アドレスが含まれています。リターンする前に、SwapInPageは無効ディスクリプタを、ページ・アドレスをもつ有効ページ・ディスクリプタに置き換えます。これでこのページが使用可能になります。

これらのルーチンは、オペレーティング・システムのメモリ管理サービスに必要な多数の機能を提供しますが、完全なメモリ管理システムを実現するには、さらに各ルーチンを補足する機能が必要です。通常、この補足機能は同じステップを逆順で実行します。GetVirtualに対応するプリミティブはReturnVirtualであり、SwapInPageに対応するのはSwapOutPageということになります。これらのプリミティブは、同様のステップ逆順で実行できるようになっています。

9.10.2 割当てルーチン

この項ではセントラル・ルーチン Vallocateを説明します。これはメモリを得るためのユーザ・プログラム・コールです。この項(および次項)では、コードに束縛されない高水準言語のシンタックスを使用します。コードは読みやすくするために、自由度が高くなっています。たとえば、コードはステータスの戻り値に対して記述文字列を割り当てます。これらの文字列は通常2進値を表わします。また、コードは空の大かっこを使用して、テーブルを操作するループ内の明白な添字を表わします。そのようなループでは、2行目の添字が明白になっています。

```
for Upper__Table__Index = 1 to 31 do
  if Upper__Table  [Upper__Table__Index]. Status = invalid  then ...
```

ここで示すコードでは、2行目は次のようになります。

```
if Upper__Table  []. Status = invalid  then ...
```

このコードはシステムのどこかで定義されることを想定したフラグ操作を使用し、ビット操作よりも複雑な操作が可能です。たとえば、無効バージンのページ・テーブル・ステータスは、ページ・ディスクリプタおよび割当て済みの未使用ページ(ページング・デバイスがページ・イメージをもたない)を示すディスクリプタ内のソフトウェア・フラグ・ビットの代わりに、無効ディスクリプタを使用して実現することができます。

Vallocateは入力パラメータとして、バイト単位で表わす所要メモリ・サイズだけを取り、ステータス情報と領域の開始を示す仮想アドレス(メモリが割り当てられている場合)を返します。このルーチンを簡略化するために、Vallocateは常にシステム・ページ・サイズの倍数を返し、16メガバイトの境界をまたがるブロックを割り当てることはありません。Vallocateはページを細分するための制御構造をインプリメントすることにより、ページの一部を割り当てることができますが、制御構造が割り当てられたページ内にあった場合は、ユーザがそれを破壊してしまうことがあります。それぞれが16メガバイトのアドレス空間をもつ下位レベル・テーブルを操作する際に、ルーチンに連続空きブロックを管理するためのコードが含まれていた場合、そのブロックは16メガバイトの境界をまたぐことができます。いったん、全領域の配置が決まると、Vallocateは連続ブロックを割り当て、最下位ブロックのアドレスを返します。

32個の上位レベル・テーブル・エントリは、ロング・ポインタ・タイプであり、それぞれ16メガバイトの仮想アドレス空間を表わします。各エントリは無効(下位ページ・テーブルをもたない)、または割当て済み(下位ページ・テーブルおよびテーブル・サイズを定義するリミット・フィールドをもつ)のいずれかです。便宜上、最初のエントリがスーパバイザ・アドレス空間をマップし、スーパバイザ保護をもちます。このルーチンが最初のエントリを修正することはありません。最初のエントリを除く31個のエントリは、ユーザ・アドレス空間として割り当てることができます。

これによく似たもので、以前に割り当てたメモリ・ブロックをリニアに拡張するルーチンを書くことができます。そのよい例がスタックです。オペレーティング・システムは、メモリの最上部(32番目の上位レベル・テーブル・エントリ)を、最上位アドレスから下位方向に成長するスタックとして割り当てることができます。タスクがいくつかの大きなスタックを必要とする場合は、下位方向への成長を示すようにソフトウェア・フラグをセットして、各スタックに16メガバイトのブロックを使用することができます。

Vallocateのロジックは次のとおりです。

1. リクエストを有効化し、要求されたページ数を計算する。
2. 各上位テーブル・エントリの下位ページ・テーブル(存在する場合)を操作して、十分な未割当てページのグループをさがす。
3. 空間が見つからなければ、下位テーブルがその最大サイズ以下かどうか、そしてブロックの末端を拡張してそれを割り当てることが可能かどうかを調べる。
4. それでも空間が見つからなければ、次の空上位テーブル・エントリを使用して、その新しい下位レベルのページ・テーブルを初期化し、そこにブロックを割り当てる。
5. 割り当てたページ・エントリをセットして、バージン・ステータス(割当て済み、無効、およびスワップ・アウトされてない)を示す。
6. ステータスを返す。ステータスがOKであれば、さらに仮想アドレスも返す。

The code for Vallocate is:

```
Vallocate (SizeInBytes, VirtualAddressReturned, Status);

/* The following are global to all routines                                         */

/* Symbolicly define the upper level pointer table                                  */

Declare Upper_Table[32] Record of
        Status=(unallocated, allocated),      /* lower table here or not            */
        Limit_Field=(0 to 4k),                /* limit for lower page table         */
        Pointer;                              /*address of lower page table if allocated */

/* Symbolicly define the lower level page table                                     */

Declare Lower_Table[0 to Limit_Field] Based Record of
        Status=(invalid_unallocated,          /*not allocated to User               */
                invalid_paged_out,            /*allocated but paged out             */
                invalid_virgin,               /*allocated but not yet used          */
                valid_in_memory),             /*allocated and in memory             */
        Pointer;                              /*physical address or disk address of page */

Declare Upper_Table_Index, Lower_Level_Index; /*table indexes                       */

Declare NumPages;                             /* number of pages required to hold request */

Status = "Out of virtual Memory";             /* default result status to this error */

if SizeInBytes > 16 megabytes then exit Vallocate;

NumPages = (SizeInBytes+PageSize-1)/PageSize; /*  Pages needed                      */

/* Scan User eligible page tables                                                   */

for Upper_Table_Index = 1 to 31 do
    If Upper_Table[].Status = allocated  then call SearchPageTable;
    If Status = "OK" then Exit Vallocate;
    end;

/* Block not found so find upper level entry unallocated and call SearchPageTable that will 'expand' */
/* the null table to hold the block.                                                 */

for Upper_Table_Index = 1 to 31
    If Upper_Table[].Status = unallocated then call SearchPageTable;

/* No more virtual space, exit leaving Status =  "out of virtual memory"            */

exit Vallocate;


Procedure SearchPageTable;

/* Scan table pointed to by upper level index to see if it can hold the block.  If not, see if  it can be  */
/* be expanded.  If successful then set flags in the page entries, set status to "OK" and User's           */
/* virtual address                                                                                          */

    Declare Maxfound;                         /* Count of consecutive free blocks found */

    Maxfound = 0;
    For Lower_Level_Index = 0 to Upper_Table[].Limit_Field
```

```
                /* count consecutive free pages until Maxfound met or not                        */
                If Lower_Table[].Status = invalid_unallocated then do
                        Maxfound = Maxfound+1;
                        if Maxfound >= NumPages then do

                                /* Found!  Now flag the page entries, update the  MC68030 and    */
                                /* return the User's virtual address                             */
                                while (Maxfound > 0) do
                                        Lower_Table[].Status = invalid_virgin;
                                        Lower_Level_Index = Lower_Level_Index-1;
                                end;

                                Status = "OK";
                                VirtualAddressReturned =
                                        Upper_Level_Index*16Meg +
                                        Lower_Level_Index*8k;
                                PLOAD (VirtualAddressReturned);
                                exit SearchPageTables;
                                end;
                        end;

        /* allocated page hit so start counting from zero again                                  */
        else Maxfound = 0;

        /* If we get here there was not room.  See if we can expand the page table to hold the new block */
        /* If so grow it and set the new page entries as virgin                                  */

        If Upper_Table[].Limit + NumPages < 4k then do
                NewLimit = Upper_Table[].Limit + NumPages;

                /* We can grow the page table!  First get area for new table                     */
                Call GetReal(4*NewLimit, NewPageTable);

                /* Now copy the first part of the old table into the new                         */
                for Lower_Table_Index = 0 to Upper_Table[].Limit
                        NewPageTable->Lower_Table[] = Lower_Table[]

                /* Return the old table and install the new table pointer                        */
                Call ReturnReal(4*Upper_Table[].Limit, Upper_Table[].Pointer);

Upper_Table[].Pointer = NewPageTable;

                /* Set returned virtual address and load it replacing the old                    */
                VirtualAddressReturned = Upper_Level_Index*16Meg + Lower_Level_Index*8k;
                PLOAD (VirtualAddressReturned)                  /* refresh MC68030               */

                /* Set all the new entries at the end to virgin status                           */
                While (Lower_Table_Index < NewLimit) do
                        Lower_Table_Index = Lower_Table_Index + 1;
                        Lower_Table[].Status = invalid_virgin;
                        end;

                /* Set OK status and return with it                                              */
                Status = "OK";
                exit SearchPageTables;
                end;

        /* cannot expand the table.  return with status unchanged (failed)                       */
        end SearchPageTables;
```

9. 10. 3 バス・エラー・ハンドラ・ルーチン

バス・エラー例外を処理するルーチンは、このオペレーティング・システム例で提供されるメモリ管理サービスの最も重要な部分です。このルーチンはページ・フォールトの妥当性を確認し、必要な処理を実行しなければなりません。また、実行タスクをアボートした条件を識別しなければなりません。PTEST命令は、エラーを発生したアクセスのアドレスとタイプを使用してテーブル・サーチを実行し、サーチ中のステータス情報を蓄積して、バス・エラーの原因を調査することができます。

PTEST命令がエラーを検出しなかった場合、そのバス・エラーはおそらく誤動作(たとえば、過渡的なメモリ障害)によるものと考えられます。オペレーティング・システムは適宜それに反応しなければなりません。

PTEST命令で実行されるテーブル・サーチは、バス・エラー・ターミネーションで終了することがあります。アドレス変換テーブルが正しく構築されていないか、メイン・メモリに故障が発生した(過渡的あるいは恒久的な故障)かのいずれかです。

スーパバイザ保護違反または書込み保護違反は、通常例外を生成するタスクがタスクのアドレス空間以外の仮想アドレス空間領域をアクセスしようとしていることを示します。オペレーティング・システムは、普通タスクを終了(アボート)させてそのようなエラーから回復します。

PTEST命令が無効ステータスを返したときには、バス・エラーはページ・フォールトであり、オペレーティング・システムは特定のタイプのページ・フォールトを識別しなければなりません。PTEST命令が返したリミット違反ビットがセットされているときは、例外を発生したタスクが未割当てのページをアクセスしようとしていたはずです。このような場合、このシステム例はタスクをアボートします。別のシステムでは、さらに多くの仮想メモリが必要なことを意味します。特に参照がタスク領域にある場合はなおさらです。

リミット違反が発生しないときには、ディスクリプタは無効です。通常、ディスクリプタは関連の情報を提供するソフトウェア・フラグをもっています。オペレーティング・システム例は、無効ディスクリプタが上位レベルまたは下位レベル・テーブルにあるかどうか調べます。ディスクリプタが上位レベル・テーブルにあれば、タスクは未割当ての仮想メモリをアクセスしようとしていたことになり、システムはそのタスクをアボートします。ディスクリプタが下位レベル・テーブルにあれば、システムはソフトウェア・フラグをチェックして無効ディスクリプタを識別します。

ソフトウェア・フラグが、ディスクリプタが未割当てページに対応していることを示しているときは、システムはタスクをアボートします。ディスクリプタがバージン・ページ(割り当てられてはいるが、まだアクセスされてない)を参照し、そのページに対する要求がリード要求の場合は、リード操作で未知のデータを読み出すため、実際にそのページは無効です。しかし、オペレーティング・システム例は要求の種類を考慮せずに、そのページに物理ページ・フレームを割り当て、ページ・テーブルにページ・ディスクリプタを書き込みます。システムによっては、バージン・ページをゼロにクリアするものもあります。

ソフトウェア・フラグが、そのページが割り当てられていて、ページング・デバイスにイメージがコピーされていることを示していれば、オペレーティング・システムはページ・フレームを割り当て、ページ・イメージをフレームの中に読み込み、そしてページ・ディスクリプタをページ・テーブルに書き込みます。もう1つ考えられる無効ディスクリプタのタイプは、仮想マシンで仮想I/Oデバイス領域を参照するものなど、特別な処理を要求するものです。

仮想ページのページ・フレームは簡単な操作で得ることができます。ただし、アイドル・ページ・フレームがないときには、システムがそれをスチールしなければなりません。スチールされたフレームのページがメモリ内で変更されている場合、システムはページ・イメージをページング・デバイ

スにセーブしなければなりません。

システムはフレームを失うタスクの変換テーブルを変更して、そのページが割り当てられ、スワップ・アウトされたことを示さなければなりません。一般に、変換テーブル・エントリはページング・デバイス上のページ・イメージのアドレスを示します。

システムがスチールするページ・フレームを選択するために使用する方法は、システムごとに大きく異なります。単純なシステムは、単に最も優先順位の低いタスクからページをスチールするだけです。さらに進んだシステムは、最も長くアクセスされなかったページ・フレームを選択します。このプロセスをエージングとよび、いくつかの方法で実行されます。1つの方法は、ページ・ディスクリプタのビットをエージング・カウンタとして使用します。オペレーティング・システムは定期的にU(使用済み)ビットを調べて、使用されていないページに対するカウントをインクリメントします。このシステムはオーバフローしたエージング・カウンタをもつページのリストを維持します。このリストのページはスチーリングに利用できます。

システムによっては、別にメモリからページ・イメージを読み出してから修正されていないページのリストを保持しています。これらのページを含むページ・フレームは、ページング・デバイス上の既存のページ・イメージが有効になったままであるため、スワップ・アウトせずにスチールできます。

ページ・スチール・ソフトウェアは、システムの多くのダイナミックスを包含することができます。それらはタスク優先順位、I/O動作、ワーキング・セットの決定、実行タスク数、スラッシング・レベル、およびその他の要素を考慮に入れることができます。

ここでとりあげたバス・エラー例外ルーチン例には、BasErrorHandlerという名前が付いています。このルーチンは、いくつかのオペレーティング・システム依存アイテムを活用しているため、Vallocateよりも一般的です。可変ポインタ VictimTaskはページ・フレームを失ったタスクから取り出したテーブルを指すことを仮定しています。この仮定は、制御ブロック・レイアウト、オペレーティング・システム例でサーチを行なって他のタスクを見つけ出す方法が定義されてないため必要です。このコードはファンクション・コード値およびリード/ライト・ステータスを省略することによってさらに簡略化でき、プログラムの基本的なロジックに影響を与えません。

```
/* Paging Bus Error Handler for example O.S.                                                   */

Procedure BusErrorHandler (BusErrAddress);

/* Global Variables to all code                                                                */

Declare TableEntry;                          /*Pointer returned by PTEST instruction           */
                                             /* pointing to the lowest level entry in the      */
                                             /* translation tables.                            */

/* Use MC68030 PTEST instruction to get fault status and table entry                           */
case PTEST (BusErrAddress,TableEntry) of

    /* Bus Error - translation table is invalid or  memory hardware problems.  Terminate the task.  */
    B:  AbortTask("Invalid table or memory hardware error");

    /* Supervisor Violation - task tried accessing restricted memory                           */
    S:  AbortTask("Attempted access of Supervisor-only memory");

    /* Write Protected - tried writing into read-only memory                                   */
    W:  AbortTask("Attempted write into read-only memory");

    /* Limit Violation - tried accessing unmapped virtual space.  This happens in our example  */
    /* O.S. when accessing within a 16 megabyte segment in User memory past what is            */
    /* currently allocated for the lower page table as determined   by the upper level limit field.  */
    L:  AbortTask("Invalid address");

    /* Invalid - pointer indicates invalid.  Must determine status.                            */
    I:  begin

            /* If upper level entry then that 16 Meg chunk of the virtual space is unallocated  */
            /* and has no page tables.                                                          */
            If TableEntry is upper level then AbortTask("Invalid address");

            /* We are at a page table entry.  Look at software flags.                           */

            /* If this page unallocated to the User then abort task                             */
            If EntryStatus=invalid_unallocated then
                                AbortTask("Invalid Address");

            /* If this page is virgin then assign to it a physical frame                        */
            if EntryStatus = invalid_virgin then do
                    GetFrame(TableEntry);              /* address returned in entry           */
                    PLOAD (BusErrAddress);             /* update MC68030 entry                */
                    exit BusErrorHandler;              /* done so continue task               */
                    end do;

            /* If this page is swapped out then read it back in */
            if EntryStatus = invalid_swapped_out then do
                    /* first get a frame to hold the new page */
                    DiskAddress = TableEntry.Pointer;  /* disk location                       */

                    GetFrame(TableEntry);              /* address returned in entry           */

                    /* Now read in the virtual page image */
                    call SwapPageIn(TableEntry,DiskAddress);
                    PLOAD (BusErrAddress);             /* update MC68851 entry                */
                    exit BusErrorHandler;              /* done so continue task               */
                    end do;
```

```
            end begin;

        /* No MC68030 status bits on.  Must be memory malfunction or RMW cycle with no          */
        /* ATC entry                                                                             */
        Otherwise:          If Stack_Frame shows RMW instruction (SSW) then
                                 /* ATC did not have descriptor loaded and MC68030 cannot        */
                                 /* search tables to load it.  Explicitly load it and allow the task to */
                                 /* continue normally                                            */

                                 Begin
                                      PLOAD (BusErrAddress); /* update ATC                       */
                                      exit BueErrorHandler;   /* done so re-execute instruction  */
                                      end Begin

                            Else: AbortTask("Memory Malfunction");

        end case;


    Procedure GetFrame(FrameTableEntry);

        /* This module returns the address of a physical frame in the passed table entry.  It obtains one */
        /* from the free frame list.  If none there it scans a queue pointing to pages that have been   */
        /* recorded as having aged by not being accessed frequently.  It first tries to find a read-only */
        /* page in the queue but if none it returns the first (oldest) entry after swapping the page out */
        /* to disk and altering the translation tables of the owning task.  If nothing in the queue it waits */
        /* for some other task to free a  frame by terminating or deallocating memory                   */

    Restart:
        if Free_Frame_Queue NOT null then
                 Dequeue first entry and return its value.

        if Aged_Frame_Queue NOT null then begin

                 /* First try to find a read-only page                                          */
                 If scanning finds read-only page then use and dequeue it
                   else dequeue the first entry (which is the oldest);

                 Find owning task and the frames current page entry;

                 /* Invalidate owning task's page                                              */
                 PFLUSH (User_Space,VictimTask.VirtualAddress);

                 /* If modified page swap it out.  SwapPageOut either gives control to other tasks */
                 /* during the I/O or copies the page returning immediately.                    */
                 If modified then call SwapPageOut(VictimTask.TableEntry);
                 /* Disk address now in Victim's page entry                                    */

                 /* Now set the old task's page status and return the frame                   */
                 VictimTask.TableEntry.Status = invalid_swapped_out;
                 return physical frame value;
                 end do;

        /* At this point we can use some other stealing method but we just wait until another task frees */
        /* a frame by terminating or freeing memory.                                              */
        call wait (Free_Frame);
        go to Restart;
```

```
        end GetFrame;

    Procedure SwapPageIn (SwapInTableEntry,DiskAddress);
        /* This procedure takes the disk address and reads the page from the paging external media   */
        /* into the physical address residing in the table entry pointer.                            */
        end SwapPageIn;

     Procedure SwapPageOut(SwapoutTableEntry);
        /* This procedure performs output on the external paging device and then replaces the        */
        /*  physical page frame address in the page entry pointer field with the disk address of the */
        /* block holding the image of the page.                                                      */
        end SwapPageOut;

    Procedure AbortTask(TerminationMsg);
        /* This procedure terminates the current task and issues a diagnostic message.               */
        end AbortTask;

   end BusErrorHandler;
```

第10章

コプロセッサ インタフェースの説明

汎用マイクロプロセッサ M68000ファミリは、広範なコンピュータ・アプリケーションを満足させる性能レベルを備えています。しかし、特定のアプリケーションに対しては、特殊目的のハードウェアを使用するほうが性能が向上する場合がよくあります。コプロセッサの概念によって、メイン・プロセッサのアーキテクチャにあまり影響を与えることなく、特定のアプリケーションに対して汎用プロセッサの機能を強化し、性能を向上させることができます。コプロセッサは、通常は汎用プロセッサによってソフトウェアで実現しなければならない特殊機能を効率よく扱うことができます。したがって、汎用メイン・プロセッサと適当なコプロセッサを組み合わせることによって、システムの処理能力を特定のアプリケーションに適応させることができます。

MC68030は本章で説明するM68000のコプロセッサ・インタフェースをサポートします。本章はMC68030にインタフェースするコプロセッサを組み込んだシステムの設計者を対象としています。

1個または複数個のモトローラのコプロセッサ(たとえば、MC68881またはMC68882浮動小数点コプロセッサ)を使用するシステムの設計者は、M68000のコプロセッサ・インタフェースについての詳細知識がなくてもかまいません。モトローラのコプロセッサは、本章で説明するインタフェースに準拠しています。通常これらは、インタフェースのサブセットをインプリメントしますが、このサブセットについてはそれぞれのコプロセッサのユーザーズ・マニュアルに記述されています。これらのコプロセッサは、各コプロセッサのユーザーズ・マニュアルに記述されているモトローラ定義の命令を実行します。

10.1 はじめに

プログラミング・モデルを見て、標準の周辺デバイスとM68000のコプロセッサの違いを明確に知っておくことが必要です。メイン・プロセッサのプログラミング・モデルは、プログラマが利用できる命令セット、レジスタ・セット、およびメモリ・マップで構成されています。M68000のコプロセッサは、M68000コプロセッサ・インタフェースとして定義されているプロトコルに従ってメイン・プロセッサと通信を行なう1デバイスまたはデバイス群です。コプロセッサのプログラミング・モデルは、周辺デバイスのそれとは異なっています。コプロセッサはメイン・プロセッサのアーキテクチャでは直接サポートされていない命令、レジスタ、およびデータ・タイプをプログラミング・モデルに追加します。コプロセッサの機能を利用するために、専用のコプロセッサ命令が用意されます。コプロセッサがサービスを提供するのに必要なメイン・プロセッサとコプロセッサとの間の対話動作は、プログラマには見えません。つまり、プログラマはメイン・プロセッサとコプロセッサ間の通信プロトコルを知っている必要はありません。なぜなら、このプロトコルはハードウェアに実装されているからです。したがって、コプロセッサはメイン・プロセッサの外部に別のハード

ウェアがあるように感じさせないで、ユーザに機能を提供することができます。

これとは対照的に、標準の周辺デバイスは一般にメイン・プロセッサのメモリ空間にマップされたインタフェース・レジスタを通してアクセスされます。プログラマは、プロセッサの標準の命令を使用して、周辺デバイスのインタフェース・レジスタにアクセスすることによって、周辺デバイスの機能を利用します。

周辺デバイスは、多くのアプリケーションにおいて概念的にはコプロセッサと等価な機能を提供することができますが、プログラマは周辺デバイスを使用するのに必要なメイン・プロセッサと周辺デバイス間の通信プロトコルをインプリメントしなければなりません。

M68000コプロセッサ・インタフェースに対して定義されている通信プロトコルについては、「10.2　コプロセッサ命令のタイプ」で説明します。M68000コプロセッサ・インタフェースをインプリメントするのに必要なアルゴリズムは、MC68030のマイクロコードで提供されており、MC68030のプログラミング・モデルでは、完全に隠されています。たとえば、浮動小数点演算はMC68030のハードウェアにはインプリメントされていませんが、MC68030とMC68881またはMC68882浮動小数点演算コプロセッサの両方を使用するシステムでは、プログラマは実際の計算がMC68881またはMC68882のハードウェアで実行されているということを意識することなく、コプロセッサに対して定義されているどの命令でも使用することができます。

10. 1. 1　インタフェース機能

M68000コプロセッサ・インタフェースには、応用性にすぐれた多数の機能が組み込まれています。物理的なコプロセッサ・インタフェースは、メイン・プロセッサの外部バスを使用しており、特殊目的の信号がないためインタフェースが簡素化されています。MC68030では、コプロセッサは非同期または同期バス転送プロトコルを使用することができます。メイン・プロセッサとコプロセッサ間の情報の転送には、標準バス・サイクルを使用するため、コプロセッサはコプロセッサ設計者が利用可能なあらゆる技術を駆使してインプリメントすることができます。コプロセッサは、VSLIデバイスとして、または別のシステム・ボードとして、あるいは別のコンピュータ・システムとしてインプリメントすることもできるのです。

メイン・プロセッサとM68000のコプロセッサは、非同期バスを通して通信できるため、同じクロック周波数で動作している必要はありません。システム設計者は、特定のシステムで最適な性能が得られるように、メイン・プロセッサおよびコプロセッサの速度を選択することができます。コプロセッサが同期バス・インタフェースを使用している場合は、すべてのコプロセッサ信号およびデータは、メイン・プロセッサのクロックと同期していなければなりません。MC68881およびMC68882浮動小数点コプロセッサは非同期バス・ハンドシェイク・プロトコルを使用します。

また、M68000コプロセッサ・インタフェースは、コプロセッサの設計を容易にします。コプロセッサの設計者は、コプロセッサ・インタフェースに適合させるだけでよく、メイン・プロセッサのアーキテクチャに関する幅広い知識は不要です。また、メイン・プロセッサはその中にコプロセッサの機能に対する特定の備えがなくても、コプロセッサと連係して動作することができます。そのため、コプロセッサを自由にインプリメントすることができます。

10. 1. 2　並行動作のサポート

M68000マイクロプロセッサ・ファミリのプログラミング・モデルは、シーケンシャルの非並行動作(non-concurrent)命令をベースにしています。これは、あるシーケンスの命令は実行される順に発生しなければならないことを意味します。一様なプログラマーズ・モデルを維持するために、コプロセッサ拡張機能もユーザ・レベルでシーケンシャルの非並行命令実行モデルを維持しなければなりません。すなわち、プログラマはある命令によって影響を受けるレジスタおよびメモリのイメー

ジは、これらのレジスタまたはメモリ・ロケーションにアクセスする、シーケンス内の次の命令が実行されるときまでに、終了しているとみなすことができます。

M68000コプロセッサ・インタフェースは、メイン・プロセッサと関連するコプロセッサ間の非並行動作に必要なすべての動作を完全にサポートします。M68000コプロセッサ・インタフェースによって、コプロセッサの並行動作が可能ですが、コプロセッサの設計者には、シーケンシャルの非並行動作の命令実行に基づくプログラミング・モデルを維持しながら、この並行性を実現する責任があります。

たとえば、コプロセッサが命令“B”が命令“A”によって変更または使用される資源を、使用または変更しないと判断した場合、命令“B”は並行して実行できます(実行用ハードウェアも利用可能な場合)。したがって、必要な命令の相互依存性およびプログラムのシーケンスが常に守られます。MC68882コプロセッサは並行して命令を実行しますが、MC68881コプロセッサは並行実行は行ないません。しかし、MC68030は、MC68881で行なわれるコプロセッサ命令の実行と並行して命令を実行することができます。

10. 1. 3　コプロセッサ命令のフォーマット

コプロセッサの命令セットは、そのコプロセッサの設計によって定義されます。コプロセッサ命令がメイン・プロセッサの命令ストリーム中に現われると、MC68030のハードウェアがそのコプロセッサとの通信を開始し、コプロセッサとの間でその命令の実行に必要な対話を調整します。プログラマはコプロセッサが定義する命令セットおよびレジスタ・セットについてだけ知っていれば、そのコプロセッサの機能を利用することができます。

M68000のコプロセッサの命令セットは、M68000の命令セットのF系列のオペレーション・ワードのサブセットを使用しています。このオペレーション・ワードは、M68000ファミリの命令の第1ワードです。F系列のオペレーション・ワードは、ビット15～12がすべて([15：12] ＝ [1111] 図10-1参照)になっており、残りのビットはコプロセッサおよび命令によって異なります。F系列のオペレーション・ワードの後にはいくらでも拡張ワードを続けることができ、それによってそのコプロセッサ命令の実行に必要な追加情報を提供します。

図10-1に示すように、F系列のオペレーション・ワードのビット9～11は、コプロセッサの識別コード(Cp-ID) をエンコードします。MC68030はコプロセッサ識別フィールドを使用して、命令が適用されるコプロセッサを示します。Cp-IDが0のF系列のオペレーション・ワードは、MC68030のコプロセッサ命令ではありません。Cp-ID(ビット9～11)とタイプ・フィールド(ビット6～8)に0が含まれている場合、この命令はMC68030のオンチップ・メモリ管理ユニットにアクセスします。Cp-IDが0でタイプ・フィールドが非ゼロの命令は、未実装命令であり、これがあるとMC68030は例外処理を開始します。MC68030はCp-IDが0の場合には、コプロセッサ・インタフェース・バス・サイクルを発生しません(MOVES命令による場合を除く)。

001～101のCp-IDコードは、現在および将来のモトローラのコプロセッサのために予約されており、110～111のCp-IDコードはユーザ定義コプロセッサのために予約されています。現在定義されているモトローラのCp-IDコードは、001がMC68881またはMC68882浮動小数点コプロセッサに対応しています。したがって、モトローラのアセンブラは、MC68881またはMC68882のコプロセッサ命令のオペレーション・コードを発生するとき、何も指定されていなかった場合はCp-IDコー

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Cp-ID			タイプ			タイプ依存					

図10-1　F系列コプロセッサ命令のオペレーション・ワード

ド001を使用します。

コプロセッサ命令のオペレーション・ワードのビット0〜8のエンコーディングは、実装されている特定の命令ごとに異なります（「10. 2　コプロセッサ命令のタイプ」を参照)。

10. 1. 4　コプロセッサ・システム・インタフェース

コプロセッサ命令を実行するのに必要なメイン・プロセッサとコプロセッサ間の通信プロトコルは、コプロセッサ内に常駐するコプロセッサ・インタフェース・レジスタとよぶ一群のインタフェース・レジスタを使用します。MC68030のハードウェアは、これらのレジスタの1つにアクセスしてコプロセッサ命令を開始します。コプロセッサは、M68000コプロセッサ・インタフェースのために定義された一組の応答プリミティブ・コードおよびフォーマット・コードを使用し、これらのレジスタを通して、メイン・プロセッサに対するステータスおよびサービス要求の通信を行ないます。コプロセッサ・インタフェース・レジスタは、メイン・プロセッサとコプロセッサ間でオペランドを受渡しするのにも使用します。コプロセッサ・インタフェースのレジスタ・セット、応答プリミティブ、およびフォーマット・コードについては、「10. 3　コプロセッサ・インタフェース・レジスタ・セット」および「10. 4　コプロセッサ応答プリミティブ」を参照してください。

10. 1. 4. 1　コプロセッサの分類

M68000コプロセッサは、バス・インタフェース機能の違いによって2つのカテゴリに分類されます。最初のカテゴリは非DMA(non-DMA)コプロセッサであり、常にバス・スレーブとして動作するコプロセッサで構成されています。第2のカテゴリはDMAコプロセッサであり、コプロセッサ・インタフェースを通してメイン・プロセッサと通信しながらバス・スレーブとして動作する一方、バス・マスタとして動作し、システム・バスを直接制御することもできます。

コプロセッサの動作に、利用可能なバス帯域幅の大部分が必要ない場合、あるいは直接メイン・プロセッサによって満足させることができない特別な要件が必要な場合、そのプロセッサは非DMAコプロセッサとしてインプリメントするのが効果的です。非DMAコプロセッサは、常にバス・スレーブとして動作するため、コプロセッサが必要とする外部バス関連の機能は、すべてメイン・プロセッサが実行します。メイン・プロセッサは、まず該当するコプロセッサ・インタフェース・レジスタ(CIR)からオペランドを読み出し、ついでファンクション・コード・ラインで適切なアドレス空間を指定し、そのオペランドを指定された実効アドレスに書き込むことによって、コプロセッサからオペランドを転送します。同様に、メイン・プロセッサはオペランドを指定された実効アドレスから読み出し、ついでコプロセッサ・インタフェースを使用してそのオペランドを該当するCIRに書き込むことによって、コプロセッサにオペランドを転送します。バス・スレーブとして動作しているコプロセッサのバス・インタフェース回路は、バス・マスタとして動作しているデバイスの場合より複雑ではありません。

メモリとコプロセッサ間のオペランド転送の効率を改善するために、比較的高い割合でバス帯域幅を使用する必要のあるコプロセッサや特別のバス要件をもつコプロセッサは、DMAコプロセッサとしてインプリメントできます。DMAコプロセッサはバス・マスタとして動作可能です。つまり、このコプロセッサはバスを要求し獲得してからそれを使用してDMA転送を行なうのに必要な、制御、アドレス、およびデータ信号をすべて提供します。しかし、DMAコプロセッサもM68000コプロセッサ・インタフェース・プロトコルを使用して、メイン・プロセッサの情報やサービスが必要になると、やはりバス・スレーブとして動作しなければなりません。

10. 1. 4. 2　プロセッサ−コプロセッサ間インタフェース

図10-2は非同期の非DMA　M68000コプロセッサ・インタフェースで使用する信号のブロック・

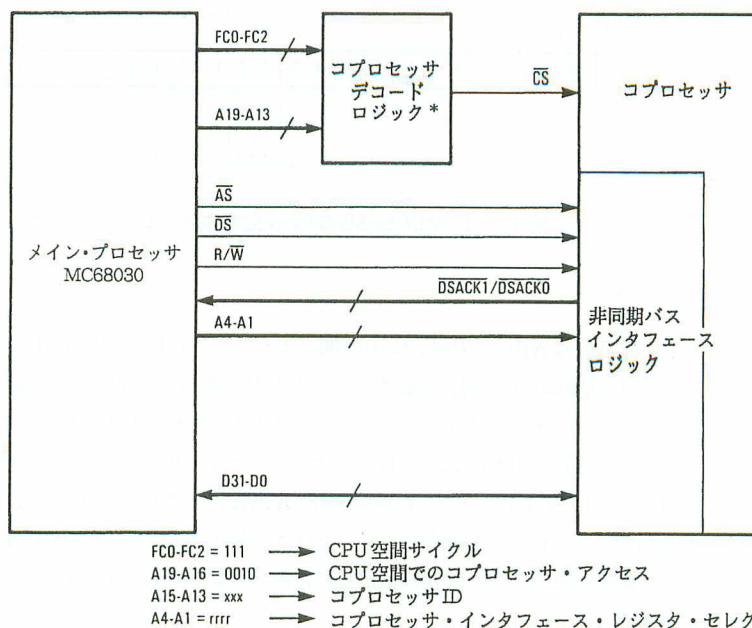



図10-2　非同期の非DMAタイプM68000コプロセッサ・インタフェースで使用する信号

レベルでのダイヤグラムです。同期インタフェースはよく似ています。アドレス・バスの信号A13-A15上のCp-IDを、他のアドレス信号とともに使用してコプロセッサを選択することができるため、システム設計者は同じタイプのコプロセッサをいくつも使用して、それぞれに固有のCp-IDを割り当てることができます。

MC68030は標準の非同期バス・サイクルを使用して、コプロセッサ・インタフェース・レジスタ・セット内のレジスタにアクセスします。したがって、コプロセッサが自分のインタフェース・レジスタのために組み込むバス・インタフェースは、MC68030のアドレス、データ、および制御信号のタイミングを満足していなければなりません。MC68030のリードおよびライトのバス・サイクルに関するタイミング情報は、図13-5から図13-8に示してあります。MC68030はコプロセッサ(CPU空間)バス・サイクル中は、バースト操作を要求せず、またコプロセッサ(CPU空間)バス・サイクル中は、データ・リードまたはライトを内部でキャッシュしません。MC68030 のバス操作の詳細については、「第7章　バス操作」を参照してください。

コプロセッサ命令の実行時に、MC68030はCPU空間バス・サイクルを実行し、コプロセッサのインタフェース・レジスタ・セットにアクセスします。MC68030は、3つのファンクション・コード出力を“H”(FC2：FC0＝111)にドライブして、CPU空間バス・サイクルを示します。コプロセッサ・インタフェース・レジスタ・セットは、周辺デバイスのインタフェース・レジスタ・セットが通常データ空間にマップされるのと同様に、CPU空間にマップされます。コプロセッサへのアクセス中に、MC68030のファンクション・コード・ラインおよびアドレス・バスにエンコードされている情報を使用して、アクセス中のコプロセッサへのチップ・セレクト信号を発生します。他のアドレス・ラインは、インタフェース・セット内のレジスタを選択します。コプロセッサへのアクセス中に、MC68030のファンクション・コードおよびアドレス・バスにエンコードされる情報を図10-3に示します。

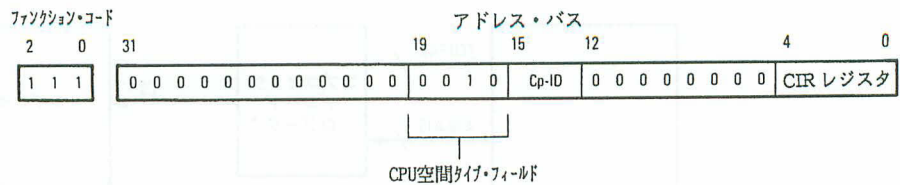



図10-3　MC68030のCPU空間のアドレス・エンコーディング

アドレス信号A16-A19は、CPU空間バス・サイクル中のCPU空間サイクルのタイプを指定します。MC68030に対して現在定義されているCPU 空間サイクルのタイプは、割込みアクノリッジ、ブレークポイント・アクノリッジ、およびコプロセッサ・アクセス・サイクルです。CPU空間タイプ＄2(A19：A16＝0010)はコプロセッサ・アクセス・サイクルを示します。

MC68030のアドレス・バスの信号A13-A15は、アクセスされているコプロセッサのコプロセッサ識別コード(Cp-ID)を指定します。このコードはコプロセッサ命令のオペレーション・ワード(図10-1 参照)のビット9-11から抽出され、各コプロセッサのアクセス中にアドレス・バス上に出力されます。

したがって、MC68030のファンクション・コード信号およびアドレス・バスのビットA13-A19をデコードすることによって、任意のコプロセッサの固有のチップ・セレクト信号を発生することができます。ファンクション・コード信号およびA16-A19は、コプロセッサへのアクセスを示し、A13-A15は7つのコプロセッサ(001から111)のどれにアクセスしているかを示します。MC68030のアドレス・バスのビットA20-A31およびA5-A12は、コプロセッサへのアクセス中は常に0です。

MC68010はMOVES命令を使用してCPU空間でのコプロセッサ・アクセス・サイクルをエミュレートすることができます。

10. 1. 4. 3　コプロセッサ・インタフェース・レジスタ(CIR)の選択

図10-4に、コプロセッサ・アクセス中にメイン・プロセッサのCPUアドレス空間の固有領域をアドレス指定する、MC68030のアドレス・バス上の値を示します。MC68030アドレス・バスの信号A0-A4は、アクセスされているコプロセッサ・インタフェース・レジスタ(CIR)を選択します。M68000コプロセッサ・インタフェースのレジスタ・マップを図10-5に示します。個々のレジスタの詳細については、「10. 3　コプロセッサ・インタフェース・レジスタ・セット」を参照してください。

10. 2　コプロセッサ命令のタイプ

M68000コプロセッサ・インタフェースは4つのカテゴリのコプロセッサ命令、すなわち、汎用、条件付き、コンテキスト・セーブおよびコンテキスト・リストアをサポートしています。

カテゴリ名は、そのカテゴリのコプロセッサ命令が供給する操作のタイプを示します。また、この命令のカテゴリは、コプロセッサ命令を起動し、命令を完全に実行させるのに必要なメイン・プロセッサとコプロセッサ間の通信プロトコルを開始するために、MC68030がアクセスするコプロセッサ・インタフェース・レジスタも決定します。汎用命令および条件付き命令のカテゴリに入る命令を実行している間、コプロセッサはM68000コプロセッサ・インタフェースのために定義されているコプロセッサ応答プリミティブ・コードを使用して、メイン・プロセッサからのサービスを要求したり、メイン・プロセッサに対してステータスを提示したりすることができます。コンテキスト・セーブおよびコンテキスト・リストアのカテゴリの命令を実行している間、コプロセッサは

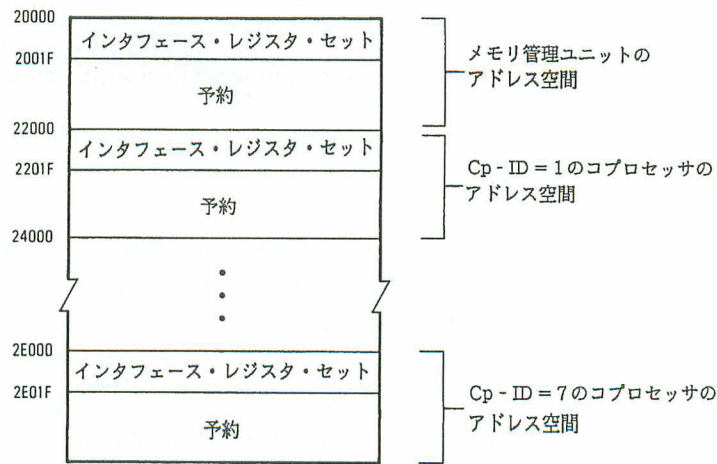



図10-4　MC68030のCPU空間のコプロセッサ・アドレス・マップ

	31	15　0
00	応答*	制御*
04	セーブ*	リストア*
08	オペランド・ワード	コマンド*
0C	(予約)	条件*
10	オペランド*	
14	レジスタ・セレクト	(予約)
18	命令アドレス	
1C	オペランド・アドレス	

図10-5　コプロセッサ・インタフェース・レジスタ・セットのマップ

M68000コプロセッサ・インタフェース用に定義されているコプロセッサ・フォーマット・コードを使用して、自分のステータスをメイン・プロセッサに提示します。

10. 2. 1　コプロセッサの汎用命令

汎用コプロセッサ命令カテゴリには、個々のコプロセッサに対して定義されているデータ処理命令および他の汎用命令があります。

10. 2. 1. 1　フォーマット

図10-6に汎用タイプの命令のフォーマットを示します。

説明の都合上、ここではすべての汎用命令にcpGENというニーモニックを使用しています。特定の汎用命令のニーモニックは、通常それが実行する操作のタイプおよび適用されるコプロセッサを示唆するようになっています。コプロセッサ命令を表わすのに使用する実際のニーモニックおよびシンタックスは、そのコプロセッサ命令のオブジェクト・コードを生成するアセンブラまたはコンパイラのシンタックスによって決まります。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Cp-ID			0	0	0	実効アドレス					
コプロセッサ・コマンド															
オプションの実効アドレスまたはコプロセッサ定義拡張ワード															

図10-6　コプロセッサ汎用命令フォーマット(cpGEN)

汎用タイプのコプロセッサ命令は、必ず最低2ワードで構成されています。この命令の第1ワードはF系列のオペレーション・コード(ビット［15：12］＝1111)です。

F系列のオペレーション・コードのCp-IDフィールドは、コプロセッサへのアクセス中に、システム内のどのコプロセッサがそのコプロセッサ命令を実行するかを示します。Cp-IDは、コプロセッサ・インタフェース・レジスタ(「10. 1. 4. 2　プロセッサ-コプロセッサ間インタフェース」参照)へのアクセス中に、アドレス・ラインA13-A15に出力されます。

ビット[8：6]＝000 でその命令が汎用命令カテゴリのものであることを示します。F系列のオペレーション・コードのビット0－5を使用して、標準のM68000の実効アドレス指示フィールド(「2. 5 実効アドレス・エンコーディングの概要」参照)をエンコードすることができます。cpGEN命令の実行中、コプロセッサはコプロセッサ応答プリミティブを使用して、MC68030にその命令に必要な実効アドレス計算を実行するよう要求することができます。MC68030はF系列オペレーション・コードの実効アドレス指示フィールドを使用して、実効アドレッシング・モードを決めます。コプロセッサが実効アドレス計算を要求しなかった場合は、ビット0－5にはどんな値があってもかまいません(“don't care”)。

汎用タイプの命令の第2ワードは、コプロセッサのコマンド・ワードです。メイン・プロセッサはこのコマンド・ワードをコマンド用CIRに書き込んで、コプロセッサによる命令の実行を開始します。

汎用のコプロセッサ命令の中には、コプロセッサ・コマンド・ワードの次にオプションでいくつかの拡張ワードを含むものがあります。これらのワードは、コプロセッサ命令の実行に必要な追加情報を提供します。たとえば、コプロセッサがコプロセッサ命令の実行中に、MC68030に対して実効アドレスの計算を要求した場合、その計算に必要な情報は実効アドレス拡張ワードとして命令のフォーマットの中に含まれていなければなりません。

10. 2. 1. 2　プロトコル

cpGEN命令は、図10-7に示すプロトコルに従って実行されます。メイン・プロセッサは、命令のコマンド・ワードをコマンド用CIRに書き込むことによって、コプロセッサとの通信を開始します。

次に、コプロセッサはそのコマンド・ワードをデコードして、cpGEN命令の処理を開始します。コプロセッサ・コマンド・ワードの解釈は、そのコプロセッサの設計によって規定されており、MC68030はデコードは行ないません。

コプロセッサは命令を実行している間、応答用CIRに適当なコプロセッサ応答プリミティブ・コードを置くことによって、メイン・プロセッサに対してサービスを要求し、ステータスを提示することができます。メイン・プロセッサはコマンド用CIRに書き込んだのち、応答用CIRを読み出してそれに適宜応答します。コプロセッサは、命令の実行を終了するか、あるいは命令を実行するためにメイン・プロセッサからサービスを受ける必要がなくなると、その旨を応答してプロセッサを解放します。そうすると、メイン・プロセッサは命令ストリームの次の命令の実行に移ります。しかし、トレース例外が保留されていた場合、MC68030はコプロセッサがcpGEN命令に関連するすべての処理の終了を示すまで、コプロセッサとの通信を終了しません(「10. 5. 2. 5　トレース例外」参照)。

図10-7に示すコプロセッサ・インタフェース・プロトコルによって、コプロセッサは汎用カテゴ

メイン・プロセッサ		コプロセッサ
M1　F系列のオペレーション・ワードを認識する。		
M2　コプロセッサ・コマンド・ワードをコマンド用CIRに書き込む。	→	C1　コマンド・ワードをデコードし、コマンドの実行を開始する。
		C2　(メイン・プロセッサのサービスが要求される間)、次のステップ1) および2) を実行する。 1)　応答用CIR に適当な応答プリミティブ・コードを置くことによって、サービスを要求する。 2)　メイン・プロセッサからサービスを受ける。
M3　応答用CIRからコプロセッサ応答プリミティブ・コードを読み出す。 1)　応答プリミティブが要求するサービスを実行する。 2)　(コプロセッサ応答プリミティブが "Come Again" (再帰要求)を示していれば)M3へ戻る(注1参照)。	↔	
		C3　応答用CIRで "No Come Again"(再帰不要)を示す。
		C4　コマンドの実行を終了する。
		C5　応答用CIRで "処理終了"を示す。
M4　次の命令の実行に移る(注2参照)。		

注：1. "Come Again"は、コプロセッサがメイン・プロセッサのサービスをさらに要求していることを示します。
　　2. 次の命令は、この時点で走査用PCが指すオペレーション・ワードになります。MC68030の走査用PCについては、「10. 4. 1　走査用PC」を参照してください。

図10-7　汎用のカテゴリの命令に対するコプロセッサ・インタフェース・プロトコル

リの命令の動作を定義することができます。つまり、メイン・プロセッサは命令のコマンド・ワードをコマンド用CIRに書き込むことによって命令の実行を開始し、応答用CIRを読み出して次の行動を決定するのです。コプロセッサ命令の実行は、コプロセッサの内部動作およびメイン・プロセッサからのサービスを要求する応答プリミティブを使用して定義されます。この命令プロトコルによって、汎用命令カテゴリの中で広範囲な操作を実行することができます。

10. 2. 2　条件付きコプロセッサ命令

条件付き命令カテゴリは、コプロセッサ動作に基づいてプログラムの制御を行ないます。コプロセッサは条件を評価し、メイン・プロセッサに真/偽の回答を返します。メイン・プロセッサは、この真/偽の応答に基づいて命令の実行を終了します。

条件付きカテゴリの命令を組み込むことにより、メイン・プロセッサおよびコプロセッサのハードウェアを効果的に使用することができます。命令に対して指定されている条件は、コプロセッサ動作に関連しているため、コプロセッサによって評価されます。しかし、条件評価に続く命令終了は直接メイン・プロセッサの動作に関連しています。フローの変更、バイトの設定、あるいはTRAP操作は、メイン・プロセッサのアーキテクチャの中にその命令セットに対する操作が組み込まれているため、メイン・プロセッサが実行します。

図10-8に条件付きカテゴリのコプロセッサ命令のためのプロトコルを示します。メイン・プロセッサは、条件用CIRに条件選択コードを書き込むことによって、このカテゴリの命令の実行を開始します。コプロセッサは条件選択コードをデコードして、評価すべき条件を知ります。コプロセッサは応答プリミティブを使用して、メイン・プロセッサにその条件に必要なサービスを提供するよう要求することができます。条件評価が終了すると、コプロセッサは応答用CIRにヌル(Null) プリミティブ(「10. 4. 4　ヌル・プリミティブ」参照)を置いて、メイン・プロセッサに真/偽の回答を返します。メイン・プロセッサは、コプロセッサからの条件評価の結果を受け取ると、そのコプロセッサ命令の実行を終了します。

メイン・プロセッサ	コプロセッサ
M1　F系列のオペレーション・ワードを認識する。	
M2　コプロセッサの条件選択コードを条件用CIRに書き込む。→	C1　条件選択コードをデコードして、条件の評価を開始する。
	C2　(メイン・プロセッサのサービスを必要とする間)、次のステップ1) および2) を実行する。 1)　応答用CIRに適当な応答プリミティブ・コードを置くことによってサービスを要求する。 2)　メイン・プロセッサからサービスを受ける。
M3　応答用CIRからコプロセッサ応答プリミティブ・コード↔を読み出す。 1)　応答プリミティブが要求するサービスを実行する。 2)　コプロセッサ応答プリミティブが "Come Again" (再帰要求)を示していればM3へ戻る(注1参照)。	
	C3　条件評価を終了する。
	C4　応答用CIRに真/偽の条件結果とともに"No Come Again" (再帰不要)ステータスを返す。
M4　次の命令の実行に移る(注2参照)。	

注：1. ヌル・プリミティブを除き、"Come Again" のプリミティブ属性が可能なすべてのコプロセッサ応答プリミティブは、条件付きのカテゴリの命令の実行中に使用したときは、必ず "Come Again" を示さなければなりません。これらのプリミティブの1つで "Come Again" 属性が示されなかった場合、メイン・プロセッサはプロトコル違反の例外処理(「10. 5. 2. 1　プロトコル違反」参照)を開始します。

図10-8　条件付きカテゴリの命令に対するコプロセッサ・インタフェース・プロトコル

10. 2. 2. 1　コプロセッサ条件分岐命令

条件付き命令カテゴリには、2種類のM68000ファミリ分岐命令のフォーマットがあります。これらの命令は、コプロセッサの動作に関連する条件に基づいて分岐します。これらの命令は、M68000ファミリの命令セットの条件分岐命令と同様に実行されます。

10. 2. 2. 1. 1　フォーマット　図10-9に、ワード長のディスプレースメントを提供するコプロセッサ条件分岐命令のフォーマットを示します。また、図10-10にロング・ワード・ディスプレースメントを含む命令のフォーマットを示します。コプロセッサ条件分岐命令の第1ワードは、F系列のオペレーション・ワードです。ビット[15：12] ＝1111およびビット[11：9]には、その条件を評価するコプロセッサの識別コードが含まれています。ビット[8：6]の値は、分岐命令のワードまたはロング・ワード・ディスプレースメント・フォーマットを識別し、それぞれcpBcc.WまたはcpBcc.Lニーモニックで指定されます。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Cp-ID			0	1	0	条件選択コード					
オプションのコプロセッサ定義拡張ワード															
ディスプレースメント															

図10-9　コプロセッサ条件付き命令での分岐(cpBcc. W)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Cp-ID			0	1	1	条件選択コード					
オプションのコプロセッサ定義拡張ワード															
ディスプレースメント——上位															
ディスプレースメント——下位															

図10-10　コプロセッサ条件付き命令での分岐(cpBcc. L)

F系列のオペレーション・ワードのビット[0-5]は、コプロセッサ条件選択フィールドです。MC68030がオペレーション・ワード全体を条件用CIRに書き込むと、コプロセッサが分岐命令の実行を開始します。コプロセッサはビット[0-5]を使用して、評価すべき条件を知ります。

コプロセッサが条件を評価するための追加情報を必要とする場合、分岐命令フォーマットの拡張ワードの中にこの情報を含めておくことができます。F系列のオペレーション・ワードに続く拡張ワード数は、そのコプロセッサの設計によって決められます。cpBcc命令フォーマットの最後のワードには、分岐の条件が満足されたときの分岐先のアドレスを計算するためにメイン・プロセッサが使用するディスプレースメントが含まれています。

10. 2. 2. 1. 2　プロトコル　図10-8にcpBcc.LおよびcpBcc.W命令のプロトコルを示します。メイン・プロセッサは、F系列のオペレーション・ワードを条件用CIRに書き込んで、コプロセッサに条件選択コードを転送することによって、その命令を開始します。メイン・プロセッサは、応答用CIRを読み出して次に行なう行動を決定します。コプロセッサは応答プリミティブを返して、条件の評価に必要なサービスを要求することができます。コプロセッサが「偽」の条件評価の回答を返した場合、メイン・プロセッサは命令ストリームの次の命令を実行します。コプロセッサが「真」の条件評価の回答を返した場合、プロセッサはMC68030の走査PC(「10. 4. 1　走査用PC」参照)にディスプレースメントを加算し、メイン・プロセッサが実行する次の命令のアドレスを求めます。走査PCはアドレスが計算されるとき、命令ストリームのディスプレースメントの第1ワードのロケーションを指していなければなりません。ディスプレースメントは2の補数の整数であり、16ビット・ワードまたは32ビット・ロング・ワードのいずれかです。プロセッサは、16ビットのディスプレースメントをロング・ワード値に符号拡張してからデスティネーション・アドレスの計算を行ないます。

10. 2. 2. 2　コプロセッサ条件でのセット

“コプロセッサ条件でのセット”命令は、コプロセッサが評価した条件に基づいてフラグ(データ可変バイト)のセットまたはリセットを行ないます。この命令の動作は、M68000ファミリの命令セットのScc命令の動作に類似しています。Scc命令およびcpScc命令は直接的にプログラムのフローを変更することはありませんが、プログラムのフローに影響を与えるフラグをセットするのによく使用します。

10. 2. 2. 2. 1　フォーマット　図10-11にcpSccのニーモニックで表わされる“コプロセッサ条件でのセット”命令のフォーマットを示します。

cpScc命令の第1ワードは、F系列のオペレーション・ワードです。このワードはビット[9-11]にCp-IDフィールドを含みビット[8:6]が001となっており、cpScc命令を示しています。F系列のオペレーション・ワードの下位6ビットは、M68000ファミリの実効アドレッシング・モード(「2. 5　実効アドレス・エンコーディングの概要」参照)をエンコードするために使用します。

cpScc命令の第2ワードは、ビット[0-5]にコプロセッサの条件選択コードを含んでいます。このワードのビット[6-15]は、モトローラによって予約されていますので、将来のM68000製品との互換性を確保するために0にしておいてください。このワードを条件用CIRに書き込んでcpScc命令を開始します。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Cp-ID			0	0	1	実効アドレス					
										条件選択コード					
オプションのコプロセッサ定義拡張ワード															
オプションの実効アドレス拡張ワード(0-5ワード)															

図10-11　コプロセッサ条件でのセット(cpScc)

コプロセッサが条件を評価するために追加情報を必要とする場合、命令に拡張ワードを含めて、その情報を供給することができます。これらの拡張ワードはコプロセッサの条件選択フィールドを含むワードに続き、その数はコプロセッサの設計によって決まります。

cpScc命令フォーマットの最後の部分は、0〜5個の実効アドレス拡張ワードを含んでいます。これらのワードには、F系列オペレーション・ワードのビット[0-5]で指定される実効アドレスの計算に必要な追加情報があります。

10. 2. 2. 2. 2 プロトコル　図10-8にcpScc命令のプロトコルを示します。MC68030は、F系列オペレーション・ワードの次のワードを条件用CIRに書き込むことによって、コプロセッサに条件選択コードを転送します。次にメイン・プロセッサは、応答用CIRを読み出して次に行なうべき動作を決定します。コプロセッサは応答プリミティブを返して、その条件を評価するのに必要なサービスを要求することができます。cpScc命令の動作は、コプロセッサからメイン・プロセッサに返される条件評価の回答によって異なります。コプロセッサが「偽」の評価結果を返した場合、メイン・プロセッサはF系列のオペレーション・ワードのビット[0-5]で指定される実効アドレスを計算し、そのアドレスにあるバイトをFALSE(全ビットがクリア)に設定します。コプロセッサが「真」の評価結果を返した場合、メイン・プロセッサは実効アドレスにあるバイトをTRUE(全ビットが1にセット)に設定します。

10. 2. 2. 3 コプロセッサ条件テスト・デクリメント分岐

“コプロセッサ条件テスト・デクリメント分岐”命令の操作は、MC68000ファミリの命令セットのDBcc命令に類似しています。この操作はコプロセッサによる条件評価結果、およびメイン・プロセッサ内のループ・カウンタに基づいて実行され、多くの高級言語で使用されているDO-UNTIL構造を実現するのに好都合です。

10. 2. 2. 3. 1 フォーマット　図10-12にDBccのニーモニックで表わされる“コプロセッサ条件テスト・デクリメント分岐”命令のフォーマットを示します。

cpDBcc命令の第1ワードはF系列のオペレーション・ワードです。このワードはビット[9-11]にCp-IDフィールドを含み、ビット[8:3]は001001となっていてcpDBcc命令を示します。この操作ワードのビット[0:2]は、命令実行中にループ・カウンタとして使用するメイン・プロセッサのデータ・レジスタを指定します。

cpDBcc命令の第2ワードには、ビット[0-5]にコプロセッサ条件選択コードが含まれています。ビット[6-15]は将来のM68000製品との互換性を維持するために0になっていなければなりません。このワードは条件用CIRに書き込まれると、コプロセッサはcpDBcc命令の実行を開始します。

コプロセッサが条件を評価するために追加情報を必要とする場合、cpDBcc命令はその情報を拡張ワードに含めることができます。これらの拡張ワードは、cpDBcc命令フォーマットでコプロセッサの条件選択フィールドを含むワードに続いています。

命令の最後のワードには、cpDBcc命令に対するディスプレースメントがあります。このディスプレースメントは、デスティネーション・アドレスの計算に使用するときは、ロング・ワードに符号拡張される2の補数形式の16ビット値です。

<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td colspan="3">Cp-ID</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td colspan="3">実効アドレス</td></tr>
<tr><td colspan="10">(予約)</td><td colspan="6">条件選択コード</td></tr>
<tr><td colspan="16">オプションのコプロセッサ定義拡張ワード</td></tr>
<tr><td colspan="16">ディスプレースメント</td></tr>
</table>

図10-12 “コプロセッサ条件のテスト、デクリメントおよび分岐”命令のフォーマット(cpDBcc)

10. 2. 2. 3. 2　プロトコル　図10-8にcpDBcc命令のプロトコルを示します。MC68030はオペレーション・ワードの次のワードを条件用CIRに書き込むことによって、コプロセッサに条件選択コードを転送します。ついでメイン・プロセッサは、応答用CIRを読み出して次に行なうべき動作を決定します。コプロセッサは応答プリミティブを使用して、その条件を評価するのに必要なサービスを要求します。コプロセッサが「真」の評価結果を返した場合、メイン・プロセッサは命令ストリーム内の次の命令を実行します。コプロセッサが「偽」の評価結果を返した場合、メイン・プロセッサはF系列のオペレーション・ワードのビット［0-2］で指定されるレジスタの下位ワードをデクリメントします。デクリメントした後のレジスタの内容がマイナス1(−1)であった場合、メイン・プロセッサは命令ストリーム内の次の命令を実行します。デクリメント後のレジスタの内容がマイナス1(−1)以外の場合、メイン・プロセッサはデスティネーション・アドレスに分岐して命令の実行を継続します。

MC68030は走査PC(「10. 4. 1　走査用PC」参照)にディスプレースメントを加算し、次の命令のアドレスを求めます。走査PCはデスティネーション・アドレスが計算されるときには、命令ストリームの16ビット・ディスプレースメントを指していなければなりません。

10. 2. 2. 4　コプロセッサ条件でのトラップ

“コプロセッサ条件でのトラップ”命令によって、プログラマはコプロセッサ動作に関連する条件に基づいて例外処理を開始させることができます。

10. 2. 2. 4. 1　フォーマット　図10-13にcpTRAPccのニーモニックで表わされる“コプロセッサ条件でのトラップ”命令のフォーマットを示します。

F系列オペレーション・ワードはビット［9-11］にCP-IDフィールド、ビット［8：3］に001111をもち、cpTRAPcc命令を示します。cpTRAPccF系列オペレーション・ワードは、命令フォーマットに含まれているオプションのオペランド・ワード数を指定します。この命令のフォーマットに含めることができるオペランド数は、0、1、または2個です。

cpTRAPcc命令フォーマットの第2ワードには、ビット［0-5］にコプロセッサ条件選択コードがあります。ビット［6-15］は将来のM68000製品との互換性を維持するために0になっていなければなりません。コプロセッサの条件用CIRにこのワードを書き込むと、コプロセッサはcpTRAPcc命令の実行を開始します。

コプロセッサが条件を評価するために追加情報を必要とする場合、命令はその情報を拡張ワードに含めることができます。これらの拡張ワードは、cpTRAPcc命令フォーマットでコプロセッサの条件選択フィールドを含むワードに続いています。

cpTRAPccのF系列オペレーション・ワードのオペランド・ワードがコプロセッサ定義の拡張ワードに続きます。これらのオペランド・ワードは、MC68030によって明示的に使用されませんが、cpTRAPcc例外処理ルーチンが参照する情報の保持に使用することができます。F系列操作ワードのビット［0-2］の有効なエンコーディングおよび対応するオペランド・ワードを表10-1に示します。これらのビットの他のエンコーディングは、cpTRAPcc命令では無効です。

<table>
<tr><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td colspan="3">Cp-ID</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td colspan="3">オペレーション・モード</td></tr>
<tr><td colspan="10">(予約)</td><td colspan="6">条件選択コード</td></tr>
<tr><td colspan="16">オプションのコプロセッサ定義拡張ワード</td></tr>
<tr><td colspan="16">オプションのワード</td></tr>
<tr><td colspan="16">またはロング・ワード・オペランド</td></tr>
</table>

図10-13　コプロセッサ条件でのトラップ(cpTRAPcc)

10. 2. 2. 4. 2 プロトコル　　図10-8にcpTRAPcc命令のプロトコルを示します。MC68030はオペレーション・ワードの次のワードを条件用CIRに書き込むことによって、コプロセッサに条件選択コードを転送します。メイン・プロセッサは、応答用CIRを読み出して次に行なうべき動作を決定します。コプロセッサは応答プリミティブを使用して、その条件を評価するのに必要なサービスを要求することができます。コプロセッサが「真」の評価結果を返した場合、メイン・プロセッサはcpTRAPcc例外(「10. 5. 2. 4 cpTRAPcc命令トラップ」参照)に対する例外処理を開始します。コプロセッサが「偽」の評価結果を返した場合、メイン・プロセッサは命令ストリームの次の命令を実行します。

表10-1 cpTRAPccのオペレーションモードのエンコーディング

Opモード	命令フォーマットのオプションのワード数
010	1
011	2
100	0

10. 2. 3 コプロセッサのコンテキスト・セーブおよびコンテキスト・リストア

M68000コプロセッサ・インタフェースのコプロセッサのコンテキスト・セーブおよびコンテキスト・リストア命令カテゴリは、マルチタスキングのプログラム環境をサポートします。マルチタスキング環境では、コプロセッサのコンテキストをそのコプロセッサ動作と非同期に変更する必要があります。つまり、コプロセッサはコンテキスト変更操作を開始するために、汎用または条件付きカテゴリの命令を実行している間、任意の時点で割込みを受ける可能性があります。

汎用および条件付き命令カテゴリとは対照的に、コンテキスト・セーブおよびコンテキスト・リストア命令カテゴリは、応答プリミティブを使用しません。これらの命令の実行中は、M68000コプロセッサ・インタフェースで定義される1組のフォーマット・コードで、メイン・プロセッサにステータス情報を知らせます。これらのコプロセッサ・フォーマット・コードについては、「10. 2. 3. 2 コプロセッサ・フォーマット・ワード」で詳しく説明しています。

10. 2. 3. 1 コプロセッサの内部状態フレーム

コンテキスト・セーブ(cpSAVE)命令およびコンテキスト・リストア(cpRESTORE)命令は、メモリとコプロセッサ間でコプロセッサ内部状態フレームを転送します。このコプロセッサの内部状態フレームは、コプロセッサの動作状態を表わしています。cpSAVE命令およびcpRESTORE命令を使用して、コプロセッサの動作を中断し、現在実行中の動作に関連するコンテキストをセーブして、新しいコンテキストでコプロセッサの動作を開始することができます。

cpSAVE命令はコプロセッサの内部状態フレームを、ロング・ワードのエントリのシーケンスとしてメモリに格納します。図10-14にコプロセッサ状態フレームのフォーマットを示します。

cpSAVE命令の実行中に、MC68030は命令のオペレーション・ワードにある情報から状態フレームの実効アドレスを計算し、その実効アドレスにフォーマット・ワードを格納します。プロセッサはコプロセッサの状態フレームを構成するロング・ワードを、実効アドレスとフォーマット・ワードの長さフィールド×4の和で指定されるアドレスから、メモリ・アドレスの降順に書き込みます。cpRESTORE命令の実行中、MC68030は状態フレーム内のフォーマット・ワードとロング・ワードを命令のオペレーション・ワードで指定されている実効アドレスから昇順に読み出します。

プロセッサはコプロセッサのフォーマット・ワードを、メモリ内の状態フレームの最下位アドレスに格納します。このワードは、cpSAVE命令およびcpRESTORE命令の両方で最初のワードとして転送されます。フォーマット・ワードの次のワードは、コプロセッサの状態フレームに関連する情報を含んでいませんが、状態フレームの中の情報のサイズを4バイトの倍数で保持する役割を果たします。フォーマット・ワードに続くエントリ(上位のアドレスにある)の数が決められます。

セーブの順序	リストアの順序	31	23	15　　　　　　　　0
0	0	フォーマット	長さ	（未使用、予約）
n	1	コプロセッサの内部情報		
n-1	2			
n-2	3			
•	•	•		
•	•	•		
•	•	•		
1	n			

図10-14　メモリ内でのコプロセッサ状態フレームのフォーマット

コプロセッサ状態フレーム内の情報は、そのコプロセッサの動作のコンテキストを記述しています。この中にはプログラムからは見えない状態情報があり、またオプションによってプログラムで調べることができる状態情報も入っています。プログラムで見えない状態情報は、内部レジスタ値またはステータス情報で、プログラムではアクセスできないがコプロセッサが中断時点から動作を再開するために必要な情報です。プログラムで見える情報には、コプロセッサのプログラミング・モデルにあり、コプロセッサ命令セットを使用して直接にアクセス可能なすべてのレジスタの内容が含まれます。cpSAVE命令でセーブされる情報の中には、プログラムで見えない情報が入っていなければなりません。プログラムで見えるコプロセッサの状態をセーブするためのcpGEN命令が用意されている場合、cpSAVE命令およびcpRESTORE命令は、セーブまたはリストアの操作での割込み待ち時間を最小にするために、プログラムで見えない情報だけを転送しなければなりません。

10. 2. 3. 2　コプロセッサ・フォーマット・ワード

コプロセッサはcpSAVE命令およびcpRESTORE命令の実行中に、コプロセッサ・フォーマット・ワードを使用してメイン・プロセッサにステータス情報を送信します。M68000コプロセッサ・インタフェースに対して定義されているフォーマット・ワードを表10-2に示します。

コプロセッサ・フォーマット・ワードの上位バイトには、メイン・プロセッサにコプロセッサ・ステータス情報を送信するのに使用するコードが含まれています。MC68030は、エンプティ/リセット(empty/reset)、ノット・レディ(not ready)、無効(invalid)、および有効(valid)の4種類のフォーマット・ワードを認識します。MC68030は予約されているフォーマット・コード($03-$0F)を無効フォーマット・ワードとして解釈します。コプロセッサ・フォーマット・ワードの下位バイトは、コプロセッサ状態フレームのサイズをバイトで(4の倍数でなければならない)指定します。こ

表10-2　コプロセッサ・フォーマット・ワードのエンコーディング

フォーマット・コード	長　さ	意　　味
00	XX	エンプティ/リセット
01	XX	ノット・レディ、再帰要求
02	XX	無効フォーマット
03-oF	XX	未定義、予約
10-FF	長さ	有効フォーマット、コプロセッサ定義

の値は有効なフォーマット・コードの場合のみ重要です(「10. 2. 3. 2. 4　有効フォーマット・ワード」参照)。

10. 2. 3. 2. 1　エンプティ/リセット・フォーマット・ワード　コプロセッサはcpSAVE命令の実行中にエンプティ/リセット・フォーマット・コードを返し、コプロセッサがユーザ固有の情報を含んでいないことを示します。すなわち、前にエンプティ/ リセット・フォーマット・コードを伴うcpRESTOREを実行してから、あるいはハードウェア・リセットを行なってから、コプロセッサ命令が実行されていないことを示しています。cpSAVE命令の開始時にメイン・プロセッサがセーブ用CIR からエンプティ/リセット・フォーマット・ワードを読み出した場合、メイン・プロセッサはcpSAVE命令の中で指定されている実効アドレスにフォーマット・ワードを格納し、次の命令の実行に移ります。

メイン・プロセッサは、cpRESTORE命令の実行中にメモリからエンプティ/リセット・フォーマット・ワードを読み出すと、そのフォーマット・ワードをリストア用CIRに書き込みます。次に、メイン・プロセッサはリストア用CIRを読み出し、コプロセッサがエンプティ/リセット・フォーマット・ワードを返した場合は、次の命令の実行に移ります。メイン・プロセッサはエンプティ/リセット・フォーマット・コードをリストア用CIRに書き込むことによって、コプロセッサを初期化することができます。コプロセッサはエンプティ/リセット・フォーマット・コードを受け取ると、現在実行中の動作を終了し、メイン・プロセッサが次のコプロセッサ命令を開始するのを待ちます。特に、エンプティ/リセット・フォーマット・ワードのcpRESTORE命令の場合、コプロセッサは以前にcpSAVE命令が実行されていれば、他のコプロセッサ命令を実行する前にこのフォーマット・ワードを返します。したがって、エンプティ/リセット状態フレームは、このフォーマット・ワードとそれに続くメモリ内の予約ワードだけで構成されます(図10-14参照)。

10. 2. 3. 2. 2　ノット・レディ・フォーマット・ワード　メイン・プロセッサがセーブ用CIRを読み出してcpSAVE命令を開始すると、コプロセッサはノット・レディ・フォーマット・ワードを返すことによって、セーブ操作を遅らせることができます。すると、メイン・プロセッサは保留されている割込みをサービスしてから、再びセーブ用CIRを読み出します。

このノット・レディ・フォーマット・ワードは、コプロセッサが自分の内部状態をセーブできる状態になるまでセーブ操作を延期します。cpSAVE命令は、汎用または条件付きコプロセッサ命令の実行を中断します。コプロセッサは、cpRESTORE命令で適切な状態がリストアされると、中断した命令を再開することができます。コプロセッサ命令の実行を終了するために、それ以上メイン・プロセッサのサービスを受ける必要がなくなった場合、その命令を終了させてセーブされる状態のサイズを小さくするほうが効率的なことがあります。コプロセッサの設計者は、メイン・プロセッサがcpSAVE命令を実行したときは、命令を終了させるのと、いったん中断して後で再開するのとどちらが効率的であるか考慮しなければなりません。

メイン・プロセッサがフォーマット・ワードをリストア用CIRに書き込むことによってcpRESTORE命令を開始したときは、コプロセッサは一般に現在実行中の命令を終了し、メイン・プロセッサが供給する状態フレームをリストアする必要があります。したがって、cpRESTORE命令の実行中は普通コプロセッサはノット・レディ・フォーマット・ワードを返しません。何らかの理由でコプロセッサがcpRESTORE操作を遅らせる場合、コプロセッサはメイン・プロセッサがリストア用CIRを読み出したときに、ノット・レディ・フォーマット・ワードを返すことができます。cpRESTORE命令の実行中に、メイン・プロセッサがリストア用CIRからノット・レディ・フォーマット・ワードを読み出した場合、メイン・プロセッサは保留されている割込みをサービスせずに、再びリストア用CIR を読み出します。

10. 2. 3. 2. 3　無効フォーマット・ワード　cpRESTORE命令を開始するために、リストア用CIRに置かれたフォーマット・ワードが有効コプロセッサ状態フレームを記述していなかったときは、コ

プロセッサはリストア用CIRに無効フォーマット・ワードを返します。メイン・プロセッサがcpRESTORE命令の実行中にこのフォーマット・ワードを読み出すと、制御用CIRにアボート・マスクを書き込み、フォーマット・エラーの例外処理を開始します。アボート・マスクの最下位2ビットは01で、最上位14ビットは定義されていません。

コプロセッサはメイン・プロセッサがcpSAVE命令を開始したとき、通常セーブ用CIRに無効フォーマット・ワードを置きません。しかし、以前に開始したcpSAVE命令またはcpRESTORE命令の実行中は、次のcpSAVE命令を開始できない場合があります。この場合、コプロセッサは別のcpSAVE命令またはcpRESTORE命令の実行中に、cpSAVE命令を開始するためにメイン・プロセッサがセーブ用CIRを読み出したときに、無効フォーマット・ワードを返すことができます。メイン・プロセッサがセーブ用CIRから無効フォーマット・ワードを読み出した場合、アボート・マスクを制御用CIRに書き込み、フォーマット・エラー例外処理(「10. 5. 1. 5　フォーマット・エラー」参照)を開始します。

10. 2. 3. 2. 4　有効フォーマット・ワード　メイン・プロセッサがcpSAVE命令の実行中に、セーブ用CIRから有効フォーマット・ワードを読み出すと、長さフィールドを使用してセーブするコプロセッサ状態フレームのサイズを決めます。cpRESTORE命令の実行中、メイン・プロセッサは命令の実効アドレスから読み出した有効フォーマット・ワードの長さフィールドを使用して、リストアするコプロセッサ状態フレームのサイズを決めます。

有効フォーマット・ワードの長さフィールドは、コプロセッサ状態フレームのサイズを表わしており、4バイトの倍数でなければなりません。cpSAVE命令またはcpRESTORE命令の実行中に、メイン・プロセッサが4バイトの倍数でないサイズの有効フォーマットの長さフィールドを検出した場合、メイン・プロセッサはアボート・マスク(「10. 2. 3. 2. 3　無効フォーマット・ワード」参照)を制御用CIRに書き込み、フォーマット・エラー例外処理を開始します。

10. 2. 3. 3　コプロセッサ・コンテキスト・セーブ命令

M68000コプロセッサ・コンテキスト・セーブ命令カテゴリには、1つの命令が定義されています。cpSAVEのニーモニックで表わされるコプロセッサ・コンテキスト・セーブ命令によって、汎用または条件付き命令カテゴリのコプロセッサ命令の実行に関係なく、コプロセッサのコンテキストをダイナミックにセーブすることができます。cpSAVE命令の実行中に、コプロセッサはコプロセッサ・フォーマット・コードを使用してメイン・プロセッサにステータス情報を知らせることができます。

10. 2. 3. 3. 1　フォーマット　図10-15にcpSAVE命令のフォーマットを示します。この命令の第1ワードはF系列のオペレーション・ワードで、その中のビット［9-11］にコプロセッサの識別コードがあり、ビット［0-5］にM68000の実効アドレス・コードが含まれています。cpSAVE命令にエンコードされている実効アドレスは、そのコプロセッサの現在のコンテキストに関連する状態フレームがセーブされるメモリのアドレスを決定するのに使用します。

cpSAVE命令に対しては制御/可変およびプリデクリメントのアドレッシング・モードが使用できます。他のアドレッシング・モードが指定されていると、MC68030はF系列エミュレータ例外処理(「10. 5. 2. 2　F系列エミュレータ例外」参照)を開始します。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Cp - ID			1	0	0	実効アドレス					
実効アドレス拡張ワード(0 - 5ワード)															

図10-15　コプロセッサのコンテキスト・セーブ命令のフォーマット(cpSAVE)

cpSAVE命令のオペレーション・ワードの次に、5個までの実効アドレス拡張ワードを含めることができます。これらのワードには、オペレーション・ワードのビット[0-5]で指定される実効アドレスを計算するために必要な追加情報が含まれています。

10. 2. 3. 3. 2 プロトコル 図10-16にコプロセッサ・コンテキスト・セーブ命令のプロトコルを示します。メイン・プロセッサは、セーブ用CIRを読み出すことによってcpSAVE命令の実行を開始します。cpSAVE命令はCIRから読出しを行なって開始する唯一のコプロセッサ命令です(他のコプロセッサ命令はすべて、CIRに書込みを行なってコプロセッサ命令の実行を開始します)。コプロセッサはセーブ用CIRにコプロセッサ・フォーマット・コードを置くことによって、コンテキスト・セーブ操作に関連するステータス情報をメイン・プロセッサに知らせます。

メイン・プロセッサがセーブ用CIRを読み出したときに、コプロセッサが現在実行中の操作を中断する用意ができていなかった場合、「ノット・レディ(Not Ready)」フォーマット・コードを返します。メイン・プロセッサは保留中の割込みがあればそれに対するサービスを実行した後、再びセーブ用CIRを読み出します。セーブ用CIRにノット・レディのフォーマット・コードを置いた後、コプロセッサは現在実行中の命令を中断するか、終了しなければなりません。

コプロセッサは実行中の命令を中断するか終了すると、コプロセッサの内部状態を示すフォーマット・コードをセーブ用CIRに置きます。メイン・プロセッサは、セーブ用CIRを読み出すと、そのフォーマット・ワードをcpSAVE命令で指定される実効アドレスに転送します。コプロセッサ・フォーマット・ワードの下位バイトは、フォーマット・ワードと関連のヌル・ワードを除いてコプロセッサから指定された実効アドレスに転送する状態情報のバイト数を指定します。状態情報のサイズが4バイトの倍数でなかった場合、MC68030はフォーマット・エラー例外処理(「10. 5. 1. 5 フォーマット・エラー」参照)を開始します。

コプロセッサとメイン・プロセッサは、オペランド用CIRを使用してコプロセッサの内部状態の転送を調整します。MC68030はコプロセッサ・フォーマット・ワードで指定されている全バイトの転送が終わるまで、オペランド用CIRを繰り返し読み出し、得た情報をメモリに書き込むことによって、コプロセッサ・コンテキスト・セーブを終了します。cpSAVE命令の後、コプロセッサはアイドル状態、すなわち、どのコプロセッサ命令も実行しない状態になっていなければなりません。

cpSAVE命令は特権命令です。メイン・プロセッサがcpSAVE命令を見つけると、ステータス・

メイン・プロセッサ	コプロセッサ
M1 F系列のオペレーション・ワードを認識する。	
M2 セーブ用CIRを読み出してcpSAVE命令を開始する。	C1 (コンテキスト・セーブ操作の開始の準備ができていない)であれば、下記のステップ1)および2)を実行する。 1) “ノット・レディ”フォーマット・コードをセーブ用CIRに置く 2) 実行中の動作を中断または終了する。
M3 (フォーマット=ノット・レディ)であれば、下記のステップ1)および2)を実行する。 ← 1) 保留されている割込みをサービスする。 2) M2へ進む。	
M3 F系列オペレーション・ワードの中で指定されている実効アドレスを評価し、その実効アドレスにフォーマット・ワードを格納する。	C2 適当なフォーマット・ワードをセーブ用CIRに置く。
M4 (フォーマット=エンプティ)であればM5へ移る、そうでなければ、フォーマット・ワードで示されるバイト数を、オペランド用CIRから実効アドレスへ転送する。	C3 オペランド用CIRを介して、フォーマット・ワードで示されるバイト数を転送する。
M5 次の命令の実行に進む。	

図10-16 コプロセッサ・コンテキスト・セーブ命令のプロトコル

レジスタのスーパバイザ・ビットをチェックし、それがスーパバイザ状態で動作しているかどうかを判定します。MC68030がユーザ特権レベルにあるとき(ステータス・レジスタのビット[13]=0)にcpSAVE命令を実行しようとした場合、コプロセッサ・インタフェース・レジスタのどれにもアクセスせずに、特権違反の例外処理を開始します(「10. 5. 2. 3　特権違反」参照)。

MC68030は、cpSAVE命令の実行中にセーブ用CIRから無効フォーマット・ワードを読み出した(または有効フォーマット・ワードの長さフィールドが4バイトの倍数でなかった)場合、フォーマット・エラーの例外処理を開始します(「10. 2. 3. 2. 3　無効フォーマット・ワード」参照)。この場合、MC68030は制御用CIRにアボート・マスク(「10. 2. 3. 2. 3　無効フォーマット・ワード」参照)を書き込んで、そのコプロセッサ命令をアボートしてから例外処理に入ります。このケースは図10-16の中には含まれていません。というのは、コプロセッサは通常cpSAVE命令のコンテキストにノット・レディまたは有効フォーマット・コードのいずれかを返すためです。しかし、コプロセッサはcpSAVEまたはcpRESTOREの命令の実行中に、これらの命令を中断することができないときに、cpSAVE命令が開始された場合は、無効フォーマット・ワードを返すことができます。

10. 2. 3. 4　コプロセッサ・コンテキスト・リストア命令

M68000コプロセッサ・コンテキスト・リストア命令カテゴリには、1つの命令が定義されています。cpRESTOREのニーモニックで表わされるコプロセッサ・コンテキスト・リストア命令は、コプロセッサが現在実行中の動作を終了して、前の状態をリストア(復元)するよう強制します。cpRESTORE命令の実行中、コプロセッサはリストア用CIRにフォーマット・コードを置くことによって、メイン・プロセッサにステータス情報を知らせます。

10. 2. 3. 4. 1　フォーマット　図10-17にcpRESTORE命令のフォーマットを示します。

この命令の第1ワードは、F系列のオペレーション・コードで、その中のビット[9-11]にコプロセッサの識別コード、そしてビット[0-5]にM68000の実効アドレス・コードが含まれています。cpRESTORE命令にエンコードされている実効アドレスは、そのコプロセッサのコンテキストが格納されているメモリの開始アドレスです。実効アドレスは、プロセッサのためにリストアされるコンテキストに関連する情報を含むコプロセッサ・フォーマット・ワードです。

cpRESTORE命令の第1ワードの後に、5つまでの実効アドレス拡張ワードを続けることができます。これらのワードには、オペレーション・ワードのビット[0-5]で指定される実効アドレスを計算するのに必要な追加情報が含まれています。

プリデクリメント・アドレッシング・モードを除くすべてのメモリ・アドレッシング・モードが使用できます。無効な実効アドレスがエンコーディングがあると、MC68030はF系列のエミュレータ例外処理を開始します(「10. 5. 2. 2　F系列エミュレータ例外」参照)。

10. 2. 3. 4. 2　プロトコル　図10-18にコプロセッサ・コンテキスト・リストア命令のプロトコルを示します。メイン・プロセッサがcpRESTORE命令を見つけると、まずその命令の実効アドレスからコプロセッサ・フォーマット・ワードを読み出します。このフォーマット・ワードには、フォーマット・コードと長さフィールドが含まれています。cpRESTORE命令の実行中、メイン・プロセッサは長さフィールドのコピーを保持して、cpRESTORE命令の実行中にコプロセッサに転送するバイト数を決定し、コプロセッサのコンテキスト・リストアを開始させるために、そのフォー

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Cp-ID			1	0	1	実効アドレス					
実効アドレス拡張ワード(0-5ワード)															

図10-17　コプロセッサのコンテキスト・リストア命令のフォーマット(cpRESTORE)

メイン・プロセッサ		コプロセッサ
M1 F系列のオペレーション・ワードを認識する。		
M2 オペレーション・ワードで指定される実効アドレスからコプロセッサ・フォーマット・コードを読み出す。		
M3 コプロセッサ・フォーマット・ワードをリストア用CIRに書き込む	→	C1 実行中の動作を終了し、フォーマット・ワードを評価する。
M4 リストア用CIRを読み出す。	↔	C2 (無効フォーマット)の場合は、リストア用CIRに無効フォーマット・コードを置く。
M5 (フォーマット＝無効フォーマット)であれば、アボート・コード$0001を制御用CIRに書き込み、フォーマット・エラー例外処理を開始する(注1参照)。		
M6 (フォーマット＝エンプティ/リセット) であれば、M7へ移る。 そうでなければ、フォーマット・ワードで指定されるバイト数をオペランド用CIRへ転送する(注2参照)。		C3 (有効フォーマット)の場合は、オペランド用CIRを介して、フォーマット・ワードで示されるバイト数を受け取る。
M7 次の命令の実行に移る。		

注：1.「10. 5. 1. 5　フォーマット・エラー」参照。
2. MC68030はM2サイクル中に、メモリから読み出しオペランドCIRへ書き込むべきバイト数を決定するために、フォーマットの中で「長さ」フィールドを読み出します。

図10-18　コプロセッサ・コンテキスト・リストア命令のプロトコル

マット・ワードをリストア用CIRに書き込みます。

コプロセッサは、リストア用CIRにフォーマット・ワードを受け取ったら、現在実行中の動作を終了して、そのフォーマット・ワードを評価しなければなりません。フォーマット・ワードがコプロセッサの設計によって決められている有効なコプロセッサ・コンテキストを示している場合、コプロセッサはリストア用CIRを通してメイン・プロセッサにそのフォーマット・ワードを返し、オペランド用CIRを通して、フォーマット・ワードで指定されたバイト数を受け取る準備をします。

メイン・プロセッサは、リストア用CIRにフォーマット・ワードを書き込んだあと、同じレジスタを読み出してcpRESTOREの対話を継続します。コプロセッサが有効なフォーマット・ワードを返した場合、メイン・プロセッサは命令の実効アドレスのフォーマット・ワードで指定されていたバイト数をオペランド用CIRに転送します。

リストア用CIRに書き込まれたフォーマット・ワードが有効なコプロセッサ状態フレームを示していなかった場合、コプロセッサは無効フォーマット・ワードをリストア用CIRに置いて、現在実行中の操作を終了します。メイン・プロセッサは、無効フォーマット・コードを受け取ると、制御用CIRにアボート・マスク(「10. 2. 3. 2. 3　無効フォーマット・ワード」参照)を書き込み、フォーマット・エラー例外処理(「10. 5. 1. 5　フォーマット・エラー」参照)を開始します。

cpRESTORE命令は特権命令です。メイン・プロセッサがcpRESTORE命令をアクセスするときは、ステータス・レジスタのスーパバイザ・ビットをチェックします。MC68030がユーザ特権レベル(ステータス・レジスタのビット[13]＝0のとき)になっているときにcpRESTORE命令を実行しようとした場合は、コプロセッサ・インタフェース・レジスタのどれにもアクセスせずに、特権違反の例外処理を開始します(「10. 5. 2. 3　特権違反」参照)。

10.3　コプロセッサ・インタフェース・レジスタ(CIR)セット

M68000コプロセッサ・インタフェースは、コプロセッサ・インタフェース・レジスタ(CIR)セットを使用してコプロセッサと交信します。このれらのコプロセッサ・インタフェース・レジスタは、コプロセッサのプログラミング・モデルとは直接関連していません。

図10-5にコプロセッサ・インタフェース・レジスタ・セットのメモリ・マップを示します。4つのすべてのカテゴリのコプロセッサ命令をもつインタフェースには、アスタリスク(＊)で示すレジスタが含まれていなければなりません。M68000コプロセッサ・インタフェースに対して定義されているコプロセッサ応答プリミティブのすべてを使用するシステムの場合、このレジスタ・モデル全体をインプリメントしなければなりません。

以下、CIRセットの各レジスタを詳細に説明します。

10.3.1　応答用CIR

コプロセッサは16ビットの応答用CIRを使用して、メイン・プロセッサに対してすべてのサービス要求(コプロセッサ応答プリミティブ)を知らせます。メイン・プロセッサは応答用CIRを読み出して、汎用および条件付き命令カテゴリの命令実行中に、コプロセッサ応答プリミティブを受け取ります。応答用CIRに対するCIRセットのベース・アドレスからのオフセットは＄00です。「10.4 コプロセッサ応答プリミティブ」を参照してください。

10.3.2　制御用CIR

メイン・プロセッサは2ビットの制御用CIRに書込みを行なって、コプロセッサから要求された例外処理を認識応答するか、コプロセッサ命令の実行をアボートします。制御用CIRに対するCIRセットのベース・アドレスからのオフセットは＄02です。制御用CIRは、そのオフセットにあるワードの下位2ビットを占有します。このワードの上位14ビットは定義されていません。図10-19にこのレジスタのフォーマットを示します。

MC68030が3種類の“例外処理要求(Take Exception)”コプロセッサ応答プリミティブの1つを受け取ると、例外アクノリッジ・マスク(102)を制御用CIRに書き込んで、そのプリミティブに認識応答し、例外アクノリッジ(XA)ビットをセットします。MC68030は制御用CIRにアボート・マスク(012)を書き込み、アボート・ビット(AB)をセットして、進行中のコプロセッサ命令をアボートします(両方のマスクの上位14ビットは定義されていません)。MC68030は次の例外状態の1つを検出すると、コプロセッサ命令をアボートします。

- 応答プリミティブを読み出した後のF系列エミュレータ例外
- スーパバイザ・チェック・プリミティブに対してスーパバイザ・チェックを実行したときの特権違反例外
- 無効フォーマット・ワードまたは無効な長さを含む有効フォーマット・ワードを受け取ったときのフォーマット・エラー例外

10.3.3　セーブ用CIR

コプロセッサはcpSAVE命令の実行中に、16ビットのセーブ用CIRを使用して、ステータス情報

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(未定義、予約)														XA	AB

図10-19　制御用CIRのフォーマット

および状態フレーム・フォーマット情報をメイン・プロセッサに知らせます。メイン・プロセッサはセーブ用CIRを読み出して、コプロセッサによるcpSAVE命令の実行を開始します。セーブ用CIRに対するCIRセットのベース・アドレスからのオフセットは$ 04です。「10. 2. 3. 2　コプロセッサ・フォーマット・ワード」を参照してください。

10. 3. 4　リストア用CIR

メイン・プロセッサはコプロセッサ・フォーマット・ワードをこの16ビットのリストア用レジスタに書き込むことによって、cpRESTORE命令を開始します。cpRESTORE命令の実行中に、コプロセッサはリストア用CIRを使用してメイン・プロセッサにステータス情報および状態フレーム・フォーマット情報を知らせます。リストア用CIRに対するCIRセットのベース・アドレスからのオフセットは$ 06です。「10. 2. 3. 2　コプロセッサ・フォーマット・ワード」を参照してください。

10. 3. 5　オペレーション・ワード用CIR

メイン・プロセッサは転送オペレーション・ワード・コプロセッサ応答プリミティブ(「10. 4. 6　オペレーション・ワード転送プリミティブ」参照)に応答して、16ビット・オペレーション・ワード用CIRに進行中の命令のF系列操作ワードを書き込みます。

オペレーション・ワード用CIRに対するCIRセットのベース・アドレスからのオフセットは$ 08です。

10. 3. 6　コマンド用CIR

メイン・プロセッサは、命令ストリームにある命令のF系列オペレーション・ワードに続く、命令コマンド・ワードをこの16ビット・コマンド・レジスタに書き込むことによって、汎用カテゴリのコプロセッサ命令を開始します。コマンド用CIRに対するCIRセットのベース・アドレスからのオフセットは$ 0Aです。

10. 3. 7　条件用CIR

メイン・プロセッサは条件選択コードを16ビットの条件用CIRに書き込むことによって、条件付きカテゴリのコプロセッサ命令を開始します。条件用CIRに対するCIRセットのベース・アドレスからのオフセットは$ 0Eです。図10-20に条件用CIRのフォーマットを示します。

10. 3. 8　オペランド用CIR

コプロセッサがオペランドの転送を要求すると、メイン・プロセッサは32ビットのオペランド用CIRを読み書きすることによって転送を実行します。オペランド用CIRに対するCIRセットのベース・アドレスからのオフセットは$ 10です。

MC68030は、このレジスタの最上位バイトに境界を揃えてオペランド用CIRとの間ですべてのオペランドを転送します。プロセッサは4バイトより長いオペランドは、ロング・ワード転送シーケンスを用いて読み書きします。オペランドのサイズが4バイトの倍数でない場合、最初のロング・ワード転送後に残っている部分は、オペランド用CIR最上位バイトに揃えられます。図10-21にMC68030がオペランド用CIRにアクセスするときに使用するオペランドのアラインメントを示しま

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(未定義、予約)										条件選択コード					

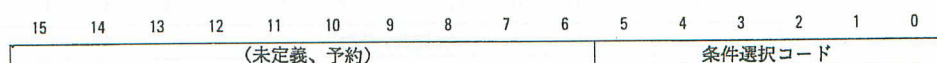



図10-20　条件用CIRのフォーマット

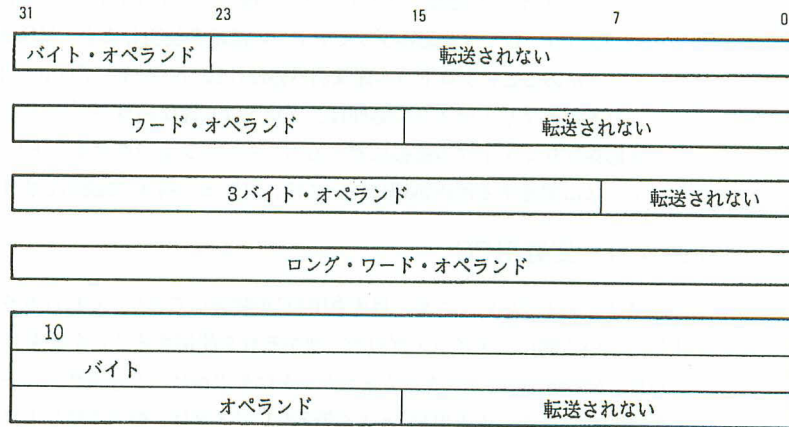



図10-21　オペランド用CIRへのアクセスのオペランドのアラインメント

す。

10. 3. 9　レジスタ選択用CIR

コプロセッサが1つまたは複数のメイン・プロセッサ・レジスタ、またはコプロセッサ・レジスタ群の転送を要求すると、メイン・プロセッサは16ビットのレジスタ選択用CIRを読み出して、転送するレジスタ数またはそのタイプを識別します。レジスタ選択用CIRに対するCIRセットのベース・アドレスからのオフセットは$ 14です。このレジスタのフォーマットは、現在それを使用しているプリミティブによって異なります。「10. 4　コプロセッサ応答プリミティブ」を参照してください。

10. 3. 10　命令アドレス用CIR

コプロセッサが現在実行中の命令のアドレスを要求した場合、メイン・プロセッサはこのアドレスを32ビットの命令アドレス用CIRに転送します。走査用PCの転送もこの命令アドレス用CIRを通して実行されます(「10. 4. 17　ステータス・レジスタおよび走査用PCの転送プリミティブ」参照)。命令アドレス用CIRに対するCIRセットのベース・アドレスからのオフセットは$ 18です。

10. 3. 11　オペランド・アドレス用CIR

コプロセッサがメイン・プロセッサとコプロセッサとの間でのオペランド・アドレスの転送を要求すると、そのアドレスはこの32ビットのオペランド・アドレス用CIRを通して転送されます。オペランド・アドレス用CIRに対するCIRセットのベース・アドレスからのオフセットは$ 1Cです。

10. 4　コプロセッサ応答プリミティブ

応答プリミティブはコプロセッサ命令の実行中に、コプロセッサがメイン・プロセッサに対して発行するプリミティブ命令です。コプロセッサはコプロセッサ応答プリミティブを使用してメイン・プロセッサにステータス情報を知らせ、サービスを要求します。コマンド用CIRに書き込まれた命令コマンド・ワード、または条件用CIRの条件選択コードに応答して、コプロセッサは応答用CIRに応答プリミティブを返します。汎用および条件付きカテゴリの命令の中で、個々の命令はコプロセッサ・ハードウェアによって、またコプロセッサ応答プリミティブが指定するサービス、そしてメイン・プロセッサが提供するサービスによって区別されます。

「10. 4. 2　コプロセッサ応答プリミティブの一般フォーマット」以下の項では、MC68030がサポートするM68000コプロセッサ応答プリミティブの詳細を説明します。MC68030が認識できない応答プリミティブがあると、プロトコル違反例外処理(「10. 5. 2. 1　プロトコル違反」参照)が開始されます。この未定義プリミティブの処理は、プロトコル違反例外ハンドラで設定されるM68000コプロセッサ応答プリミティブの拡張に対するエミュレーションをサポートします。コプロセッサ・インタフェースに関連する例外処理については、「10. 5　例外」で説明します。

10. 4. 1　走査用PC

応答プリミティブのいくつかには走査用PCが関係しており、それらの多くは要求されたサービスを実行している間に、メイン・プロセッサがそれを使用することを要求します。コプロセッサ命令の実行中、MC68030のプログラム・カウンタはその命令のF系列オペレーション・ワードのアドレスを保持しています。走査用PCとよぶ第2のレジスタは、命令の残りのワードをシーケンシャルにアドレス指定します。

メイン・プロセッサが、実行アドレスまたは分岐操作の分岐アドレスを計算するために拡張ワードを必要とする場合、走査用PCを使用して命令ストリーム内の拡張ワードをアドレス指定します。また、コプロセッサが拡張ワードの転送を要求した場合、走査用PCは転送中にその拡張ワードをアドレス指定します。プロセッサが各ワードを参照するたびに、走査用PCをインクリメントして命令ストリームの次のワードを指すようにします。1つの命令が終了すると、プロセッサは走査用PC内の値をプログラム・カウンタに転送して、次に実行する命令のオペレーション・ワードをアドレス指定します。

命令開始後、メイン・プロセッサが最初の応答プリミティブを読み出すときの走査用PCの値は、実行中の命令によって異なります。cpGEN命令の場合、走査用PCはコプロセッサ・コマンド・ワードの次のワードを指しています。cpBcc命令の場合、走査用PCは命令のF系列オペレーション・ワードの次のワードを指しています。cpScc、cpTRAPcc、cpDBccの各命令の場合、走査用PCはコプロセッサ条件指定ワードの次のワードを指しています。

コプロセッサが汎用または条件付き命令を定義するためにオプションの命令拡張ワードを使用するように設計されている場合、そのコプロセッサは命令の実行中にそれらのワードを一貫して使用し、それに従って走査用PCが更新されるようにしなければなりません。特に、汎用カテゴリの命令の実行中、コプロセッサが命令プロトコルを終了したとき、MC68030は走査用PCが次に実行する命令のオペレーション・ワードを指しているものと仮定します。条件付きカテゴリの命令の実行中、コプロセッサが命令のプロトコルを終了したとき、MC68030は走査用PCが命令フォーマットで最後にあるコプロセッサ定義拡張ワードの次のワードを指していると仮定します。

10. 4. 2　コプロセッサ応答プリミティブの一般フォーマット

M68000コプロセッサの応答プリミティブは、応答用CIRを通してメイン・プロセッサに転送される16ビット・ワードにエンコードされます。図10-22にコプロセッサ応答プリミティブのフォーマットを示します。

コプロセッサ応答プリミティブのビット［0-12］のエンコーディングは、個々のプリミティブによって異なります。しかし、ビット［13-15］はM68000コプロセッサ・インタフェースに定義され

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	DR	ファンクション					パラメータ							

図10-22　コプロセッサ応答プリミティブ

たほとんどのプリミティブに適用されるオプションの追加操作を指定します。

CAビットであるビット［15］は、メイン・プロセッサの再帰要求(come again)操作を指定します。メイン・プロセッサがこのCAビットを1にセットして、応答用CIRから応答プリミティブを読み出すと、メイン・プロセッサはそのプリミティブで示されるサービスを実行したあと、再び応答用CIRを読み出します。コプロセッサはCAビットを使用して、1つのコプロセッサ命令を実行中にメイン・プロセッサに複数の応答プリミティブを転送することができます。

PCビットのビット［4］は“プログラム・カウンタの受渡し”操作を指定します。メイン・プロセッサが応答用CIRからこのPCビットがセットされている応答プリミティブを読み出すと、メイン・プロセッサはそのプリミティブの要求に対する最初のサービスとして、すぐに自分のプログラム・カウンタの現在値を命令アドレス用CIRに渡します。プログラム・カウンタの値は、現在実行中のコプロセッサ命令のF系列オペレーション・ワードのアドレスです。PCビットは、M68000コプロセッサ・インタフェースに対して現在定義されているコプロセッサ応答プリミティブのすべてに実装されています。

未定義プリミティブ、または不当操作を要求するプリミティブがメイン・プロセッサに渡されると、メイン・プロセッサはF系列エミュレータまたはプロトコル違反のいずれかに対する例外処理を開始します(「10. 5. 2　メイン・プロセッサ検出例外」参照)。しかし、これらの応答プリミティブの1つでPCビットがセットされていた場合、メイン・プロセッサは例外処理を開始する前に、命令アドレス用CIRにプログラム・カウンタの値を渡します。

メイン・プロセッサが、メイン・プロセッサの命令と並行して実行されるcpGEN命令を開始したときには、最初にコプロセッサが返すプリミティブのPCビットは通常セットされています。メイン・プロセッサはコプロセッサから解放されると、命令ストリームの実行に進むため、コプロセッサは命令の実行に関連して起こりうる例外の処理をサポートするために、その命令のアドレスを記録しておかなければなりません。コプロセッサ命令の並行実行に関連する例外処理については、「10. 5. 1　コプロセッサが検出する例外」を参照してください。

DRビットであるビット［13］は方向ビットです。このビットはメイン・プロセッサとコプロセッサ間のオペランドの転送に適用されます。DR＝0の場合、転送はメイン・プロセッサからコプロセッサの方向に行なわれます(メイン・プロセッサの書込み)。DR＝1の場合、転送はコプロセッサからメイン・プロセッサの方向に行なわれます(メイン・プロセッサの読込み)。ある応答プリミティブが示す操作が、明示的なオペランド転送を含まない場合、このビット値は個々のプリミティブのエンコーディングによって異なります。

10. 4. 3　ビジー・プリミティブ

ビジー応答プリミティブが返されると、メイン・プロセッサはそのコプロセッサ命令を再実行します。このプリミティブは、汎用および条件付きカテゴリの命令に適用されます。図10-23にビジー・プリミティブのフォーマットを示します。

このプリミティブは、PCビットを上記のとおり使用します。

メイン・プロセッサと並行して動作できるが、書込み操作をコマンド用または条件用CIRにバッファすることができないコプロセッサが、ビジー・プリミティブを使用します。コプロセッサはcpGEN命令をメイン・プロセッサでの命令と並行して実行することができます。コプロセッサが

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	PC	1	0	0	1	0	0	0	0	0	0	0	0	0	0

図10-23　ビジー・プリミティブのフォーマット

cpGEN命令を並行して実行しているときに、メイン・プロセッサが汎用または条件付きカテゴリの命令を開始しようとした場合、コプロセッサはビジー・プリミティブを応答用CIRに置くことができます。メイン・プロセッサはこのプリミティブを読み出すと、保留されている割込みをサービス("命令実行前例外スタック・フレーム"(図10-41参照)を使用)します。

次に、プロセッサは前に開始しようとした汎用または条件付きコプロセッサ命令を再開します。

ビジー・プリミティブは、コマンドまたは条件用CIRへの書込みに対する応答によってしか使用できません。このプリミティブは、メイン・プロセッサが汎用または条件付きカテゴリの命令を開始しようとした後、最初に返されるプリミティブでなければなりません。特に、ビジー・プリミティブはプログラム可視資源をその命令で変更された後に発行してはなりません(プログラム可視資源には、コプロセッサまたはメイン・プロセッサのプログラム可視レジスタおよびメモリ内のオペランドを含み、走査用PCは含みません)。命令でプログラム可視資源を変更した後、その命令を再実行すると、プロセッサが命令を再開したときに、それらの資源の一貫性が失われてしまいます。

MC68030は、ブレークポイント操作中に発生する可能性のある特殊ケースでは、ビジー・プリミティブに対し異なった応答を行ないます(「8. 1. 12　多重例外」参照)。この特殊ケースは、ブレークポイント・アクノリッジ・サイクルがコプロセッサのF系列命令を開始したときに発生し、コプロセッサはその命令の開始に応答してビジー・プリミティブを返し、割込みは保留されます。これらの3つの条件が満たされたときは、割込み例外処理が終了した後、プロセッサはブレークポイント・アクノリッジ・サイクルを再実行します。ブレークポイントを使用してカウンタをインクリメントまたはデクリメントし、ループを通過する回数をモニタする設計の場合、このような状態では正しい結果が得られないことがあります。なぜなら、この特殊ケースによってループを1回通過する間に、複数のブレークポイント・アクノリッジ・サイクルが実行されることがあるからです。

10. 4. 4　ヌル・プリミティブ

コプロセッサのヌル応答プリミティブは、コプロセッサのステータス情報をメイン・プロセッサに知らせるのに使用します。このプリミティブは、汎用および条件付きカテゴリの命令に適用されます。図10-24にヌル・プリミティブのフォーマットを示します。

このプリミティブはCAおよびPCビットを上記のとおり使用します。

ビット［8］のIAビットは、割込みを許可するオプション操作を指定します。このビットは、MC68030がヌル・プリミティブを受け取った後、応答用CIRの再読出しを行なう前に、保留されている割込みのサービスを行なうかどうかを決定します。IAビットがセットされているときは、割込みが許可されます。

ビット［1］のPFビットは、コプロセッサの「処理終了」ステータスを示します。すなわち、PF＝1はコプロセッサがある命令に関連するすべての処理を終了したことを示しています。

ビット［0］のTFビットは、条件付きカテゴリ命令の実行中の真/偽の状態を示します。TF＝1は真の状態を示し、TF＝0は偽の状態を示します。TFビットは条件付き命令の実行中にコプロセッサが使用する、CA＝0のヌル・プリミティブに対してのみ関係します。

MC68030は、汎用または条件付きカテゴリのコプロセッサ命令のいずれを実行している場合でも、CA＝1のヌル・プリミティブを同様に処理します。

コプロセッサがヌル・プリミティブのCAとIAを1にセットした場合、メイン・プロセッサは保

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	0	0	1	0	0	IA	0	0	0	0	0	0	PF	TF

図10-24　ヌル・プリミティブのフォーマット

表10-3 “コプロセッサのヌル応答プリミティブ” のエンコーディング

CA	PC	IA	PF	TF	汎用命令	条件付き命令
×	1	×	×	×	プログラム・カウンタを命令アドレス用CIRに書き込み、PCビットをクリアして、CA、IA、PF、およびTFで指定される操作に進む。	汎用命令と同じ
1	0	0	×	×	応答用CIRを再度読み出し、保留されている割込みはサービスしない。	汎用命令と同じ
1	0	1	×	×	保留されている割込みをサービスし、応答用CIRを再度読み出す。	汎用命令と同じ
0	0	0	0	c	応答用CIRを再度読み出し(トレース保留の場合)、そうでなければ、次の命令を実行する。	メイン・プロセッサはTF＝cに基づいて命令の実行を完了する。
0	0	1	0	c	保留されている割込みをサービスし応答用CIRを再度読み出す。そうでなければ、次の命令を実行する(トレース保留の場合)。	メイン・プロセッサはTF＝cに基づいて命令の実行を完了する。
0	0	×	1	c	コプロセッサ命令が終了する。保留されている例外をサービスするか次の命令を実行する。	メイン・プロセッサはTF＝cに基づいて命令の実行を完了する。

×＝Don't Care
c＝コプロセッサ条件の評価結果によって1または0

留されている割込みをサービスし(図10-43に示す“命令実行途中での例外”スタック・フレームを使用して)、再び応答用CIR を読み出します。コプロセッサがヌル・プリミティブでCAを1にセットしIAを0にセットした場合、メイン・プロセッサは保留されている割込みがあってもサービスせず、再び応答用CIRを読み出します。

CA＝0のプリミティブは、条件付きの命令の実行中にメイン・プロセッサに対して条件の評価結果を知らせ、その命令におけるメイン・プロセッサとコプロセッサ間の対話を終了させます。メイン・プロセッサはプリミティブを受け取ると、条件付きカテゴリ・コプロセッサ命令の実行を終了します。プリミティブそのものが処理の終了を暗黙に示しているため、PFビットは条件付き命令の実行中は関係ありません。

通常、汎用カテゴリ命令実行中に、メイン・プロセッサがCA＝1でないプリミティブを読み出すと、メイン・プロセッサとコプロセッサ間の対話動作を終了します。しかし、トレース例外が保留されている場合、応答用CIRからCA＝0、PF＝1のヌル・プリミティブを読み取るまでは命令対話を終了しません(「10. 5. 2. 5　トレース例外」参照)。したがって、メイン・プロセッサはCA＝0、PF＝1のヌル・プリミティブを受け取るまで応答用CIR からの読出しを繰り返し実行し、その後、トレース例外の処理を行ないます。IA＝1のとき、メイン・プロセッサは再び応答用CIRから読出しを行なう前に、保留されている割込みのサービスを実行します。

コプロセッサは、メイン・プロセッサの命令実行と並行してcpGEN命令を実行することができ、コマンド用または条件用CIRへの書込みを1回だけバッファすることができます。このタイプのコプロセッサは、cpGEN 命令を並行して実行していて、メイン・プロセッサが別の汎用命令または条件付き命令を開始したときに、CA＝1のヌル・プリミティブを発行します。このプリミティブはコプロセッサがビジーであり、メイン・プロセッサが命令を再開せずに応答用CIRを読み出さなければならないことを示しています。このヌル・プリミティブのIAビットは、通常メイン・プロセッサがコプロセッサの汎用カテゴリ命令の終了を待っているときの割込みの待ち時間を最小にするためにセットします。

表10-3にヌル・プリミティブのエンコーディングの要約を示します。

10. 4. 5　スーパバイザ・チェック・プリミティブ

スーパバイザ・チェック・プリミティブは、コプロセッサ命令の実行中にメイン・プロセッサが

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	PC	0	0	0	1	0	0	0	0	0	0	0	0	0	0

図10-25 “スーパバイザ・チェック”プリミティブのフォーマット

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	0	0	0	1	1	1	0	0	0	0	0	0	0	0

図10-26 “オペレーション・ワード転送”プリミティブのフォーマット

スーパバイザ状態で動作しているかどうかを検証します。このプリミティブは、汎用および条件付きカテゴリの命令で使用できます。図10-25にスーパバイザ・チェック・プリミティブのフォーマットを示します。

このプリミティブは、上記の説明のとおりPCビットを使用します。ビット［15］は1で示されていますが、汎用カテゴリ命令の実行中には、このプリミティブはビット［15］の値に関係なく、同じ動作を実行します。しかし、条件付きカテゴリの命令でこのプリミティブがビット［15］＝0で発行された場合、メイン・プロセッサはプロトコル違反例外処理を開始します。

メイン・プロセッサは、応答用CIRからスーパバイザ・チェック・プリミティブを読み出すと、ステータス・レジスタのSビットの値をチェックします。S＝0(メイン・プロセッサがユーザ特権レベルで動作している)の場合、メイン・プロセッサは制御用CIRにアボート・マスク(「10. 3. 2　制御用CIR」参照)を書き込むことによって、コプロセッサ命令の実行をアボートします。次に、メイン・プロセッサは特権違反例外処理を開始します(「10. 5. 2. 3　特権違反」参照)。メイン・プロセッサがこのプリミティブを受け取ったときに、スーパバイザ・モードになっていた場合、メイン・プロセッサは再び応答用CIRを読み出します。

スーパバイザ・チェック・プリミティブによって、特権命令は汎用および条件付きカテゴリのコプロセッサ命令を定義することができます。したがって、このプリミティブは特権命令として実装されている命令に対する対話動作の中で、最初にコプロセッサが発行する必要があります。

10. 4. 6　オペレーション・ワード転送プリミティブ

オペレーション・ワード転送プリミティブは、コプロセッサにコプロセッサ命令のオペレーション・ワードのコピーを要求します。このプリミティブは、汎用または条件付きカテゴリ命令で使用できます。図10-26にオペレーション・ワード転送プリミティブのフォーマットを示します。

このプリミティブは上記の説明のとおりCAおよびPCビットを使用します。このプリミティブが条件付きカテゴリ命令の実行中にCA＝0で発行された場合、メイン・プロセッサはプロトコル違反例外処理を開始します。

メイン・プロセッサは、応答用CIRからこのプリミティブを読み出すと、現在実行中のコプロセッサ命令のF系列オペレーション・ワードを、オペレーション・ワード用CIRに転送します。走査用PCの値は、このプリミティブに影響されません。

10. 4. 7　命令ストリームからの転送プリミティブ

命令ストリームからの転送プリミティブ、命令ストリームからコプロセッサへのオペランドの転送を開始します。このプリミティブは、汎用および条件付き命令で使用することができます。図10

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	0	0	1	1	1	1	長さ							

図10-27　“命令ストリームからの転送”プリミティブのフォーマット

-27に命令ストリームからの転送プリミティブのフォーマットを示します。

このプリミティブはCAおよびPCビットを上記の説明のとおり使用します。このプリミティブを条件付き命令の実行中にCA＝0で発行した場合、メイン・プロセッサはプロトコル違反例外処理を開始します。

このプリミティブのフォーマットのビット［0-7］は、命令ストリームからコプロセッサに転送するオペランドの長さをバイト単位で指定します。この長さは、偶数バイトでなければなりません。奇数長を指定した場合、メイン・プロセッサはプロトコル違反例外処理を開始します(「10. 5. 2. 1 プロトコル違反」参照)。

このプリミティブはコプロセッサにコプロセッサ定義の拡張ワードを転送します。メイン・プロセッサは、応答用CIRからこのプリミティブを読み出すと、その長さフィールドで指定されるバイト数を命令ストリームからオペランド用CIR にコピーします。最初に転送されるワードまたはロング・ワードは、メイン・プロセッサがこのプリミティブを読み出したときに、走査用PCで指定されるロケーションにあり、走査用PCは各ワードが転送されたあとインクリメントされます。したがって、このプリミティブの実行終了時点では、走査用PCは転送されたバイトの合計個数だけインクリメントされ、転送されたワードの次のワードを指しています。メイン・プロセッサは、ロング・ワード書込みシーケンスを使用して、命令ストリームからのオペランドをオペランド用CIRに転送します。長さフィールドが4バイトの偶数倍でなかった場合、ワード書込みを使用して命令ストリームからの最後の2バイトがオペランド用CIRに転送されます。

10. 4. 8　実効アドレスの評価および転送プリミティブ

実効アドレスの評価および転送プリミティブは、コプロセッサ命令のオペレーション・ワードで指定されている実効アドレスを評価し、その結果をコプロセッサに転送します。このプリミティブは、汎用カテゴリ命令で使用できます。条件付きカテゴリの命令の実行中に、コプロセッサがこのプリミティブを発行したとすると、メイン・プロセッサはプロトコル違反例外処理を開始します。図10-28に実効アドレスの評価および転送プリミティブのフォーマットを示します。

このプリミティブは上記の説明のとおりCAおよびPCビットを使用します。

メイン・プロセッサが汎用カテゴリの命令の実行中に、このプリミティブを読み出すと、メイン・プロセッサはその命令で指定されている実効アドレスを評価します。この時点では、走査用PCには要求される実効アドレス拡張ワードの最初のアドレスが入っています。メイン・プロセッサは、それらの各拡張ワードを参照した後、走査用PCを2だけインクリメントします。実効アドレスを計算した後、結果の32ビット値がオペランド・アドレス用CIRに書き込まれます。

MC68030がこのプリミティブに応答して計算するのは可変制御アドレッシング・モードだけです。オペレーション・ワードのアドレッシング・モードが可変制御モードでない場合、メイン・プロセッサは制御用CIRに＄0001を書き込むことによってその命令をアボートし、F系列エミュレーショ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	0	0	1	0	1	0	0	0	0	0	0	0	0	0

図10-28　“実効アドレスの評価および転送”プリミティブのフォーマット

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	DR	1	0	有効EA			長さ							

図10-29 “実効アドレスの評価およびデータ転送”プリミティブのフォーマット

ン例外処理を開始します(「10. 5. 2. 2 F系列エミュレータ例外」参照)。

10. 4. 9 実効アドレスの評価およびデータの転送プリミティブ

実効アドレスの評価およびデータの転送プリミティブは、コプロセッサとコプロセッサ命令のオペレーション・ワードで指定される実効アドレスとの間で、データを転送します。このプリミティブは汎用カテゴリの命令で使用できます。コプロセッサが条件付きカテゴリの命令の実行中にこのプリミティブを発行すると、メイン・プロセッサはプロトコル違反例外処理を開始します。図10-29に実効アドレス評価およびデータ転送プリミティブのフォーマットを示します。

このプリミティブは、上記の説明のとおりCA、PC、およびDRビットを使用します。

このプリミティブのフォーマットの有効実効アドレス・フィールド(ビット [8-10])は、このプリミティブに対して有効な実効アドレス・カテゴリを指定します。この命令のオペレーション・ワードで指定されている実効アドレスが、ビット [8-10] で指定されているクラスのメンバでなかった場合、メイン・プロセッサは制御用CIR(「10. 3. 2 制御用CIR] 参照)にアボート・マスクを書き込むことによって、そのコプロセッサ命令をアボートし、F系列エミュレーション例外処理を開始します。表10-4に有効な実効アドレス・フィールドのエンコーディングを示します

プリミティブで指定されている有効な実効アドレス・フィールドと命令のオペレーション・ワードで指定されている値が一致した場合、MC68030はそのプリミティブが非可変実効アドレスへの書込みを要求していれば、プロトコル違反例外処理を開始します。

転送するオペランドのバイト数は、プリミティブ・フォーマットのビット [0-7] で指定されます。ある実効アドレッシング・モードで使用するオペランド長フィールドには、いくつかの制限があります。実効アドレスがメイン・プロセッサのレジスタの場合(レジスタ直接モード)、1、2、または4バイトのオペランド長だけが有効であり、それ以外の長さ(たとえば、0)の場合、メイン・プロセッサはプロトコル違反例外処理を開始します。メモリ・アドレッシング・モードでは、0～255バイトのオペランド長が有効です。

0～255バイトの長さは、イミディエイト・オペランドでは使用できません。イミディエイト・オペランド長は1バイトまたは偶数(256以下)でなければならず、転送方向はコプロセッサに向かっていなければなりません。そうでなかった場合、メイン・プロセッサはプロトコル違反例外処理を開始します。

メイン・プロセッサが、汎用のカテゴリ命令の実行中にこのプリミティブを受け取ると、まず命

表10-4 有効な実効アドレス・コード

000	制御可変
001	データ可変
010	メモリ可変
011	可変
100	制御
101	データ
110	メモリ
111	任意の実効アドレス（制限なし）

令のオペレーション・ワードにエンコードされている実効アドレスが、そのプリミティブで指定されている実効アドレス・カテゴリの中にあるかどうか検査します。そうであった場合、プロセッサは現在の走査用PCアドレスにある実効アドレス拡張ワードを使用して実効アドレスを計算し、1ワードを参照するたびに走査用PCを2だけインクリメントします。次に、メイン・プロセッサはプリミティブの中で指定されているバイト数を、可能であればロング・ワード転送を使用してオペランド用CIR と実効アドレスとの間で転送します。オペランド用CIR に関係する転送におけるオペランドのアラインメントの詳細については、「10. 3. 8　オペランド用CIR 」を参照してください。

オペランド転送の方向は、DRビットによって指定されます。DR＝0の場合は、実効アドレスからオペランド用CIRへの転送を要求し、DR＝1の場合はオペランド用CIRから実効アドレスへの転送を要求します。

実効アドレッシング・モードがプリデクリメント・モードを指定している場合、使用するアドレス・レジスタは転送前にオペランド・サイズだけデクリメントされます。オペランドのバイトは、デクリメントされたアドレス・レジスタで指定されるロケーションを先頭とする昇順アドレスとの間で転送されます。このモードでは、A7をアドレス・レジスタとして使用し、オペランドが1バイトの場合、A7はワード・アラインメントのスタックを維持するために、2だけデクリメントされます。

ポストインクリメントの実効アドレッシング・モードの場合、使用するアドレス・レジスタは転送後にオペランドのサイズだけインクリメントされます。オペランドのバイトは、アドレス・レジスタで指定されているロケーションを先頭とする、昇順のアドレスとの間で転送されます。このモードでは、A7をアドレス・レジスタとして使用し、オペランド長が1バイトの場合、A7はワード・アラインメントのスタックを維持するために、転送後2だけインクリメントされます。1より長い奇数長のオペランド転送で、－(A7)または(A7)＋のアドレッシング・モードを使用すると、スタック・ポインタはワード境界にアラインメントされません。

プロセッサは、ある命令の実行中にこのプリミティブが発行されるたびに、繰り返し実効アドレスの計算を行ないます。実効アドレスの計算は、要求されるアドレス・レジスタおよびデータ・レジスタの現在値を使用して行なわれます。この命令には、計算を繰り返すたびに、要求される1組の実効アドレス拡張ワードが含まれていなければなりません。プロセッサは、走査用PCの現在値で示されるロケーションからこれらの拡張ワードを読み出し、命令ストリームで各ワードが参照されるたびに走査用PCを2だけインクリメントします。

MC68030は、このプリミティブをレジスタ直接実効アドレッシング・モードを使用して、アドレス・レジスタ(A0-A7)に転送するとき、バイト・サイズおよびワード・サイズのオペランドをロング・ワードに符号拡張します。

データ・レジスタ(D0-D7)に転送されるバイトまたはワードのサイズのオペランドは、データ・レジスタの下位バイトおよびワード部分にだけ上書きします。

10. 4. 10　評価済み実効アドレスへの書込みプリミティブ

評価済み実効アドレスへの書込みプリミティブは、コプロセッサから前に計算した実効アドレスにオペランドを転送します。このプリミティブは、汎用カテゴリの命令で使用できます。コプロセッサが条件付き命令の実行中にこのプリミティブを使用した場合、メイン・プロセッサはプロトコル違反例外処理を開始します。図10-30に評価済み実効アドレスへの書込みプリミティブのフォー

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	1	0	0	0	0	0	長さ							

図10-30　“評価済み実効アドレスへの書込み”プリミティブのフォーマット

マットを示します。

このプリミティブは、上記の説明のとおりCAおよびPCビットを使用します。

プリミティブ・フォーマットのビット［0-7］は、オペランドの長さをバイト数で指定します。MC68030は0～255バイトの長さのオペランドを転送します。

メイン・プロセッサが汎用カテゴリの命令の実行中にこのプリミティブを受け取ると、オペランド用CIRからMC68030内部の一時レジスタで指定される実効アドレスにオペランドを転送します。現在の命令の前のプリミティブが実効アドレスを評価したときには、この一時レジスタには、評価された実効アドレスが入っています。評価された実効アドレスをメイン・プロセッサの一時アドレスに格納するプリミティブは、“実効アドレス評価・転送”、“実効アドレス評価およびデータ転送”、または“複数コプロセッサ・レジスタ転送”プリミティブです。命令のオペレーション・ワードで指定された実効アドレスがまだ計算されていないときに、命令でこのプリミティブを使用した場合、その書込みに使用される実効アドレスは、未定義の状態です。また、前に評価された実効アドレスが“レジスタ直接”であった場合も、このプリミティブに応答して書き込まれる値は未定義となります。

書込み操作中のファンクション・コード値は、MC68030がこのプリミティブを読み出した時点でのそのステータス・レジスタのSビットの値に応じて、スーパバイザまたはユーザ・データ空間のいずれかを指します。コプロセッサは可変実効アドレッシング・モードにおいてのみ書込みを要求する必要がありますが、MC68030はこのプリミティブが使用する実効アドレスの種類についてはチェックしません。たとえば、評価済みの実効アドレスが“プログラム・カウンタ相対”であって、MC68030がユーザ特権レベルに(ステータス・レジスタのS＝0)になっている場合、MC68030は前に計算されたプログラム・カウンタ相対アドレス(プロセッサ内の一時レジスタに保持されている32ビット値)にあるユーザ・データ空間に書き込みます。

4バイトより長いオペランドは、可能であれば4バイト(オペランド部分)単位で転送されます。メイン・プロセッサはオペランド用CIRからロング・ワード・オペランド部分を読み出し、この部分を現在の実効アドレスに転送します。転送はこの方法により、オペランドのロング・ワード部分がすべて転送されるまで、昇順メモリ・ロケーションを使用して行なわれ、その後、残りのオペランド部分が1、2、または3バイト転送を使用して転送されます。オペランド部分はMC68030の一時レジスタに入っているアドレスを先頭とする昇順アドレスを使用してメモリに格納されます。

前に評価した実効アドレス・モードがプリデクリメント・モードまたはポストインクリメント・モードの場合でも、このプリミティブの実行によって、MC68030のプログラミング・モデルのレジスタが影響を受けることはありません。前に評価された実効アドレッシング・モードがMC68030の内部アドレス・レジスタまたはデータ・レジスタを使用していた場合、実効アドレス値には前のプリミティブからの最終値が使用されます。つまり、このプリミティブは“実効アドレスの評価・転送”、“実効アドレス評価およびデータ転送”、または“複数コプロセッサ・レジスタ転送”プリミティブからの値をそのまま使用します。

次の項で述べる“アドレス取得およびデータ転送”プリミティブは、MC68030が計算した実効アドレスを置き換えません。メイン・プロセッサが、“アドレス取得およびデータ転送”プリミティブに応答して取得するアドレスは、“評価済み実効アドレスへの書込み”プリミティブが参照することはできません。

コプロセッサは、“実効アドレス評価およびデータ転送”プリミティブを発行し、その後にこのプリミティブを続けることによって、不可分の“リード・モディファイ・ライト”操作を実効することができます。この操作のバス・サイクルは、ノーマル・バス・サイクルであり、割込みおよびサイクル間での調停が可能です。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	DR	0	0	1	0	1	長さ							

図10-31　“アドレス取得およびデータ転送”プリミティブのフォーマット

10. 4. 11　アドレス取得およびデータ転送プリミティブ

アドレス取得およびデータ転送プリミティブは、コプロセッサとコプロセッサが供給するアドレス間でオペランドを転送します。このプリミティブは、汎用または条件付きカテゴリの命令で使用できます。図10-31にアドレス取得および転送プリミティブのフォーマットを示します。

このプリミティブは、上記の説明のとおりCA、PCおよびDRビットを使用します。コプロセッサが条件付き命令の実行中にCA＝0でこのプリミティブを発行した場合、メイン・プロセッサはプロトコル違反例外処理を開始します。

このプリミティブのフォーマットのビット［0-7］は、オペランドの長さを0～255で指定します。

メイン・プロセッサは、最初にオペランド・アドレス用CIRから32ビットのアドレスを読み出します。次に、一連のロング・ワードを使用して、このアドレスとオペランド用CIRとの間でオペランドを転送します。転送の方向はDRビットで決まります。プロセッサは、オペランド部分をオペランド・アドレス用CIRから読み出したアドレスを先頭にして、昇順のアドレスに対して読み書きします。オペランド長が4バイトの倍数でなかった場合、オペランドの最後の部分は1、2、または3バイトの転送を使用して転送されます。

オペランド・アドレス用CIRから読み出したアドレスとともに使用するファンクション・コードは、MC68030のステータス・レジスタのSビットの値によって、スーパバイザ空間またはユーザ空間のいずれかを示します。

10. 4. 12　スタックの先頭との間の転送プリミティブ

スタックの先頭との間の転送プリミティブは、コプロセッサと現在アクティブなメイン・プロセッサ・スタック（「2. 8. 1　システム・スタック」参照）の先頭との間でオペランドを転送します。このプリミティブは汎用および条件付きカテゴリの命令で使用できます。図10-32に“スタックの先頭との転送”プリミティブのフォーマットを示します。

このプリミティブは上記の説明のとおりCA、PC、およびDRビットを使用します。コプロセッサがこのプリミティブを条件付きカテゴリの命令の実行中にCA＝0で発行した場合、メイン・プロセッサはプロトコル違反例外処理を開始します。

このプリミティブのフォーマットのビット［0-7］は、転送するオペランドの長さをバイト単位で指定します。オペランドの長さは1、、2、または4バイトが許されます。長さフィールドがこれ以外の値になっていた場合、メイン・プロセッサはプロトコル違反例外処理を開始します。

DR＝0の場合、メイン・プロセッサは現在アクティブなシステム・スタックからオペランド用CIRにオペランドを転送します。したがって、この転送で暗黙に使用される実効アドレス・モードは(A7)＋のアドレッシング・モードです。オペランドの長さが1バイトの場合、転送後にスタック・ポイン

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	DR	0	1	1	1	0	長さ							

図10-32　“スタックの先頭との転送”プリミティブのフォーマット

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	DR	0	1	1	0	0	0	0	0	0	D/A	レジスタ		

図10-33 “単一メイン・プロセッサ・レジスタの転送”プリミティブのフォーマット

タはスタックのワード・アラインメントを維持するために、2だけインクリメントされます。

DR＝1の場合、メイン・プロセッサはオペランド用CIRから現在アクティブなスタックにオペランドを転送します。したがって、この転送で暗黙に使用される実効アドレス・モードは－(A7)のアドレッシング・モードです。オペランドの長さが1バイトの場合、転送前にスタック・ポインタはスタックのワード・アラインメントを維持するために、2だけデクリメントされます。

10. 4. 13 単独のメイン・プロセッサ・レジスタの転送プリミティブ

単一のメイン・プロセッサ・レジスタの転送プリミティブは、メイン・プロセッサのデータ・レジスタまたはアドレス・レジスタの1つとコプロセッサ間でオペランドを転送します。このプリミティブは、汎用または条件付きカテゴリの命令で使用できます。図10-33に“単独のメイン・プロセッサ・レジスタの転送”プリミティブのフォーマットを示します。

このプリミティブは、上記の説明のとおりCA、PC、およびDRビットを使用します。コプロセッサが条件付きカテゴリの命令の実行中にCA＝0で発行された場合、メイン・プロセッサはプロトコル違反例外処理を開始します。

ビット［3］のD/Aビットは、プリミティブがデータ・レジスタまたはアドレス・レジスタのいずれを転送するかを示します。D/A＝0 はデータ・レジスタを示し、D/A＝1 はアドレス・レジスタを示します。ビット［2-0］にはレジスタ番号が含まれています。

DR＝0の場合、メイン・プロセッサは指定されたレジスタにあるロング・ワード・オペランドをオペランド用CIRに転送します。DR＝1の場合、メイン・プロセッサはオペランド用CIRからロング・ワード・オペランドを読み出し、それを指定されたデータ・レジスタまたはアドレス・レジスタに転送します。

10. 4. 14 メイン・プロセッサ制御レジスタの転送プリミティブ

メイン・プロセッサ制御レジスタの転送プリミティブは、制御レジスタの1つとコプロセッサ間でロング・ワード・オペランドを転送します。このプリミティブは、汎用または条件付きカテゴリの命令で使用できます。図10-34に“メイン・プロセッサ制御レジスタの転送”プリミティブのフォーマットを示します。このプリミティブは、上記の説明のとおり、CA、PCおよびDRビットを使用します。コプロセッサがこのプリミティブを条件付きカテゴリの命令の実行中にCA＝0で発行した場合、メイン・プロセッサはプロトコル違反例外処理を開始します。

メイン・プロセッサがこのプリミティブを受け取ると、まずレジスタ選択用CIRから制御レジスタ選択コードを読み出します。このコードはどのメイン・プロセッサ制御レジスタを転送するかを

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	DR	0	1	1	0	1	0	0	0	0	0	0	0	0

図10-34 “メイン・プロセッサ制御レジスタの転送”プリミティブのフォーマット

表10-5　メイン・プロセッサ制御レジスタの選択コード

16進コード	制御レジスタ
x000	ソース・ファンクション・コード(SFC)レジスタ
x001	デスティネーション・ファンクション・コード(DFC)レジスタ
x002	キャッシュ制御レジスタ(CACR)
x800	ユーザ・スタック・ポインタ(USP)
x801	ベクタ・ベース・レジスタ(VBR)
x802	キャッシュ・アドレス・レジスタ(CAAR)
x803	マスタ・スタック・ポインタ(MSP)
x804	割込みスタック・ポインタ(ISP)
他のコードはすべてプロトコル違反例外を発生します。	

決定します。表10-5に有効な制御レジスタ選択コードを示します。制御レジスタ選択コードが有効でなかった場合、MC68030はプロトコル違反例外処理を開始します(「10. 5. 2. 1　プロトコル違反」参照)。

レジスタ選択用CIRから有効なコードを読み出した後、DR＝0の場合、メイン・プロセッサは指定された制御レジスタからのロング・ワードをオペランド用CIRに書き込みます。DR＝1の場合、メイン・プロセッサはオペランド用CIRからロング・ワード・オペランドを読み出して、それを指定された制御レジスタに入れます。

10. 4. 15　複数のメイン・プロセッサ・レジスタの転送プリミティブ

複数のメイン・プロセッサ・レジスタの転送プリミティブは、1個または複数のデータ・レジスタまたはアドレス・レジスタとコプロセッサ間でロング・ワードのオペランドを転送します。このプリミティブは汎用または条件付きカテゴリの命令で使用できます。図10-35に“複数のメイン・プロセッサ・レジスタの転送”プリミティブのフォーマットを示します。

このプリミティブは上記の説明のとおり、CA、PCおよびDRビットを使用します。コプロセッサが条件付きカテゴリ命令の実行中にCA＝0で発行された場合、メイン・プロセッサはプロトコル違反例外処理を開始します。

メイン・プロセッサはこのプリミティブを受け取ると、まずレジスタ選択用CIRから16ビットのレジスタ選択マスクを読み出します。レジスタ選択マスクのフォーマットを図10-36に示します。レジスタ選択マスクで1にセットされているビットに対応するレジスタが転送されます。選択されたレジスタはD0-D7、そしてA0-A7の順に転送されます。

DR＝0の場合、メイン・プロセッサは、ロング・ワード転送シーケンスを使用してレジスタ選択マスクの中で指定された各レジスタの内容をオペランド用CIRに書き込みます。DR＝1の場合、メ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	DR	0	0	1	1	0	0	0	0	0	0	0	0	0

図10-35　“複数のメイン・プロセッサ・レジスタの転送”プリミティブのフォーマット

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A7	A6	A5	A4	A3	A2	A1	A0	D7	D6	D5	D4	D3	D2	D1	D0

図10-36　レジスタ選択マスクのフォーマット

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	DR	0	0	0	0	1	長さ							

図10-37 “複数のコプロセッサ・レジスタの転送”プリミティブのフォーマット

イン・プロセッサはオペランド用CIRから、レジスタ選択マスクで指定された各レジスタにロング・ワードを読み出します。レジスタはDRビットで示される転送方向とは関係なく、同じ順序で転送されます。

10. 4. 16 複数のコプロセッサ・レジスタの転送プリミティブ

複数のコプロセッサ・レジスタの転送プリミティブは、コプロセッサ命令で指定されている実効アドレスとコプロセッサとの間で、0～16個のオペランドを転送します。このプリミティブは、汎用カテゴリの命令で使用できます。コプロセッサが条件付き命令の実行中にこのプリミティブを発行した場合、メイン・プロセッサはプロトコル違反例外処理を開始します。図10-37に“複数のコプロセッサ・レジスタの転送”プリミティブのフォーマットを示します。

このプリミティブは上記の説明のとおり、CA、PC、およびDRビットを使用します。

このプリミティブのフォーマットのビット[7-0]は、転送する各オペランドの長さをバイト単位で示します。オペランドの長さは偶数バイトでなければならず、オペランドの長さが奇数だった場合、MC68030はプロトコル違反の例外処理を開始します(「10. 5. 2. 1 プロトコル違反」参照)。

メイン・プロセッサがこのプリミティブを読み出すと、コプロセッサ命令のオペレーション・ワードで指定される実効アドレスを計算します。このプリミティブを応答用CIRから読み出したとき、走査用PCは実効アドレス拡張ワードの最初のものを指しているはずであり、走査用PCは実効アドレスの計算中に、各拡張ワードを参照するたびに2だけインクリメントされます。実効アドレスからコプロセッサに転送する場合(DR＝0)、制御アドレッシング・モードおよびポストインクリメント・アドレッシング・モードが有効です。コプロセッサから実効アドレスに転送する場合(DR＝1)、制御可変およびプリデクリメントのアドレッシング・モードが有効です。不当なアドレッシング・モードを使用すると、MC68030は制御用CIRにアボート・マスク(「10. 3. 2 制御用CIR」参照)を書き込むことによってその命令をアボートし、F系列エミュレータ例外処理を開始します(「10. 5. 2. 2 F系列エミュレータ例外」参照)。

実効アドレスを計算した後、MC68030はレジスタ選択用CIRから16ビットのレジスタ選択マスクを読み出します。コプロセッサはレジスタ選択マスクを使用して、転送するオペランド数を指定します。MC68030は単にレジスタ選択マスクにある1の数をカウントして、転送するオペランド数を決定します。したがって、レジスタ選択マスクの中の1の順序は、メイン・プロセッサの動作には無関係であり、このプリミティブに応答してメイン・プロセッサは16個までのオペランドを転送することができます。転送されるオペランドの合計バイト数は、転送されるオペランド数とこのプリミティブのフォーマットのビット[0-7]に指定される各オペランドの長さの積になります。

DR＝1の場合、メイン・プロセッサはレジスタ選択マスクで指定されている数のオペランドをオペランド用CIRから読み出し、可能であればロング・オペランド転送を使用して、それらのオペランドを命令で指定されている実効アドレスに書き込みます。DR＝0の場合、メイン・プロセッサはレジスタ選択マスクで指定されている数のオペランドを読み出して、それらをオペランド用CIRに書き込みます。

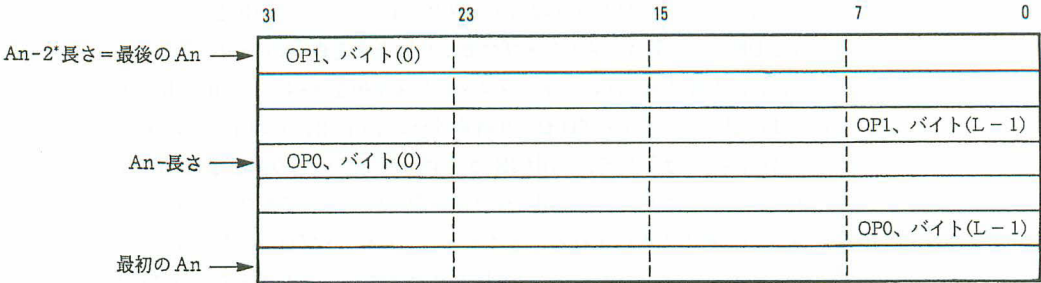
制御アドレッシング・モードの場合、オペランドは昇順アドレスを使用して、メモリとの間で転送されます。ポストインクリメント・アドレッシング・モードの場合、オペランドは昇順アドレスでメモリから読み出され、使用するアドレス・レジスタは各オペランドが転送された後、そのオペ

注：OP0、バイト(0)は最初にメモリに書き込まれるバイト。
　　OP0、バイト(L-1)は第1オペランドで最後にメモリに書き込まれるバイト。
　　OP1、バイト(0)は第2オペランドで最後にメモリに書き込まれるバイト。
　　OP1、バイト(L-1)は最後にメモリに書き込まれるバイト。

図10-38　－(An)への転送時のメモリでのオペランドのフォーマット

ランドのサイズだけインクリメントされます。(An)＋のアドレッシング・モードで使用するアドレス・レジスタは、このプリミティブの実行中に転送されたバイトの総数だけインクリメントされます。

プリデクリメント・アドレッシング・モードの場合、オペランドはアドレスの降順にメモリに書き込まれますが、各オペランド内のバイトはアドレスの昇順にメモリに書き込まれます。一例として、図10-38にロング・ワード幅のメモリに2つの12バイト・オペランドを、－(An)のアドレッシング・モードを使用して、コプロセッサから実効アドレスに転送する場合のフォーマットを示します。プロセッサはオペランドが転送される前に、アドレス・レジスタをそのオペランドのサイズだけデクリメントします。プロセッサは各オペランドのバイトを、アドレスの昇順にメモリに書き込みます。アドレス・レジスタは、転送が完了したときには、転送された総バイト数だけデクリメントされています。MC68030は可能なかぎりロング・ワード転送を使用してデータを転送します。

10. 4. 17　ステータス・レジスタおよび走査用PCの転送プリミティブ

ステータス・レジスタおよび走査用PCの転送プリミティブは、コプロセッサとメイン・プロセッサのステータス・レジスタ間で値を転送します。オプションによって、走査用PCも転送を行なうことができます。このプリミティブは汎用カテゴリの命令で使用できます。コプロセッサが条件付きカテゴリの命令の実行中にこのプリミティブを発行した場合、メイン・プロセッサはプロトコル違反例外処理を実行します。図10-39に“ステータス・レジスタおよび走査用PCの転送”プリミティブのフォーマットを示します。

このプリミティブは上記の説明のとおり、CA、PC、およびDRビットを使用します。

ビット［8］のSPビットは、走査用PCオプションを選択します。SP＝1の場合、プリミティブは走査用PCとステータス・レジスタの両方を転送します。SP＝0の場合、ステータス・レジスタだけを転送します。

SP＝0でDR＝0の場合、メイン・プロセッサは16ビットのステータス・レジスタ値をオペランド用CIRに書き込みます。SP＝0でDR＝1の場合、メイン・プロセッサはオペランド用CIRから

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	DR	0	0	0	1	SP	0	0	0	0	0	0	0	0

図10-39　“ステータス・レジスタおよび走査用PCの転送”プリミティブのフォーマット

16ビット値をメイン・プロセッサのステータス・レジスタに読み出します。

SP＝1でDR＝0の場合、メイン・プロセッサはまず走査用PCの中のロング・ワード値を命令アドレス用CIRに書き込み、次にステータス・レジスタ値をオペランド用CIRに書き込みます。SP＝1でDR＝1の場合、メイン・プロセッサはオペランド用CIRから16ビット値をステータス・レジスタに読み出し、次に命令アドレス用CIR からロング・ワードの値を走査用PCに読み出します。

このプリミティブによって、汎用カテゴリの命令は新しい値をステータス・レジスタまたは走査用PC、あるいはその両方に入れることによって、メイン・プロセッサのプログラムの流れを変えることができます。ステータス・レジスタにアクセスすることによって、コプロセッサはメイン・プロセッサのコンディション・コード、スーパバイザ・ステータス、トレース・モード、アクティブ・スタックの選択、および割込みマスク・レベルを確認し操作することができます。

このプリミティブがDR＝1(メイン・プロセッサへの転送)で発行されたときには、MC68030はメイン・プロセッサがプリフェッチした現在の走査用PCの指すロケーション以降の命令ワードを廃棄します。

次に、MC68030はステータス・レジスタのSビットで示されるアドレス空間の走査用PCアドレスから命令パイプを再充てんします。

コプロセッサ命令の実行が開始されたときに、MC68030が“フローの変化でのトレース”(ステータス・レジスタのT1/T0＝01)で動作しており、このプリミティブがDR＝1(コプロセッサからメイン・プロセッサ)で発行された場合、MC68030はトレース例外の準備を行ないます。トレース例外はコプロセッサがその命令に関連するすべての処理を完了したことを知らせたときに発生します。ステータス・レジスタをメイン・プロセッサに転送することによって起こるトレース・モードの変化は、次の命令を実行することにより有効になります。

10. 4. 18　命令実行前の例外処理要求プリミティブ

このプリミティブは、コプロセッサから供給された例外ベクタ番号および命令実行前の例外スタック・フレーム・フォーマットを使用して例外処理を開始します。このプリミティブは、汎用または条件付きカテゴリの命令で使用することができます。図10-40に“命令実行前の例外処理要求”プリミティブのフォーマットを示します。

このプリミティブは上記の説明のとおりPCビットを使用します。ビット［0-7］は、例外処理を開始するためにメイン・プロセッサが使用する例外ベクタ番号があります。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	PC	0	1	1	1	0	0	ベクタ番号							

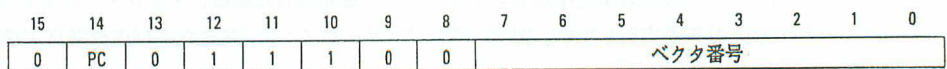



図10-40　“命令実行前の例外処理要求”プリミティブのフォーマット

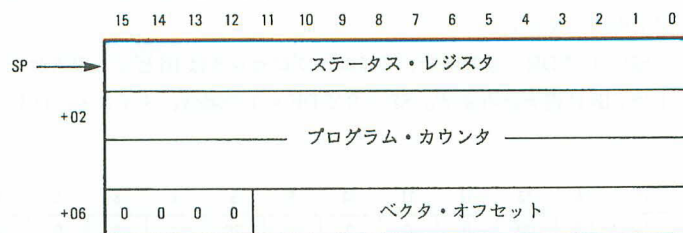



図10-41　MC68030の“命令実行前の例外”用スタック・フレーム

メイン・プロセッサがこのプリミティブを受け取ると、制御用CIRに例外アクノリッジ・マスクを書き込むことによって、コプロセッサの例外処理要求を認識応答します。次に、MC68030は「8.1　例外処理シーケンス」で説明する例外処理に入ります。この例外のベクタ番号は、プリミティブのビット［0-7］から取り出されたもので、MC68030は図10-41に示す4ワードのスタック・フレーム・フォーマットを使用します。

このスタック・フレームにセーブされるプログラム・カウンタ値は、このプリミティブを受け取ったときに実行されていたコプロセッサ命令のF系列オペレーション・ワードのアドレスです。したがって、例外ハンドラ・ルーチンがスタック・フレームを変更しなかった場合、RTE命令によってMC68030はコプロセッサ命令を再実行することになります。

このプリミティブは、コプロセッサがコプロセッサ命令を開始するために、コマンド用CIRまたは条件用CIRのいずれかに書き込まれた値を認識しないときに使用することができます。

また、このプリミティブは、プログラム可視資源が命令の操作によって変更される前に、コプロセッサ命令の中で例外が発生した場合にも使用することができます。このプリミティブは、プログラム可視資源がプロセッサ命令で変更されていた場合は、その命令の実行中には使用できません。そうでないと、MC68030は例外処理から復帰するとその命令を再開するため、再開された命令が以前に変更された資源を以前とは一貫性のない状態で受け取ることになるためです。

“命令実行前の例外処理要求”プリミティブの最も重要な用途の1つは、メイン・プロセッサの命令実行と並行して実行されていたcpGEN命令での例外状態を知らせることです。コプロセッサがcpGEN命令を実行するのにメイン・プロセッサのサービスを必要としなくなった場合、そして命令が並行して実行されていることがプログラミング・モデルからは見えない場合、コプロセッサはCA＝0でこのプリミティブを発行することによって、メイン・プロセッサを解放することができます。したがって、メイン・プロセッサは通常命令ストリームの次の命令を実行し、コプロセッサはメイン・プロセッサの動作と並行して自分の動作を実行します。コプロセッサが命令を並行して実行している間に例外が発生した場合、メイン・プロセッサが次の汎用命令または条件付き命令を実行しようとするまで、その例外は処理されません。メイン・プロセッサは、コマンド用CIRまたは条件用CIRに書込みを行なって、汎用または条件付き命令を開始した後、応答用CIRを読み出します。この時点で、コプロセッサは“命令実行前の例外処理要求”プリミティブを返すことができます。このプロトコルによって、メイン・プロセッサは前の並行に実行されるコプロセッサ命令に関連した例外処理を開始し、その後、例外が発生したコプロセッサ命令の再開を行なうことができます。コプロセッサは、メイン・プロセッサと並行して実行でき、例外の回復をサポートするすべての汎用命令のアドレスを記録しておく必要があります。次のコプロセッサ命令が開始されるまで、例外はレポートされないため、通常プロセッサは例外が発生した時点でコプロセッサがどの命令を実行していたかを知るために、その命令のアドレスを必要とします。コプロセッサはメイン・プロセッサを解放する前に使用するプリミティブの1つにPC＝1を設定することによって、命令アドレスを記録することができます。

10.4.19　命令実行途中での例外処理要求プリミティブ

このプリミティブは、コプロセッサから供給された例外ベクタおよび“命令実行途中での例外”スタック・フレーム・フォーマットを使用して例外処理を開始します。このプリミティブは、汎用または条件付きカテゴリの命令で使用できます。図10-42に“命令実行途中での例外処理要求”プリミティブのフォーマットを示します。

このプリミティブは上記の説明のとおりPCビットを使用します。ビット［70］には例外処理を開始するためにメイン・プロセッサが使用する例外ベクタ番号があります。

メイン・プロセッサがこのプリミティブを受け取ると、制御用CIR に例外アクノリッジ・マスク

(「10. 3. 2　制御用CIR」参照)を書き込むことによって、コプロセッサの例外要求に認識応答します。次に、MC68030は「8. 1　例外処理シーケンス」で説明する例外処理を実行します。この例外のベクタ番号は、このプリミティブのビット［0-7］から取り出され、MC68030は図10-43に示す10ワードのスタック・フレーム・フォーマットを使用します。

このスタック・フレームにセーブされるプログラム・カウンタ値は、プリミティブを受け取ったときに実行していたコプロセッサ命令のオペレーション・ワードのアドレスです。走査用PCフィールドには、このプリミティブを受け取ったときのMC68030の走査用PCの値が入っています。現在実行中の命令が、例外処理要求プリミティブの前に、実効アドレスの評価を行なっていなかった場合、スタック・フレームの実効アドレス・フィールドの値は未定義です。

コプロセッサはこのプリミティブを使用して、メイン・プロセッサとの対話動作中の例外に対する例外処理を要求することができます。例外ハンドラがスタック・フレームを変更しなかった場合、MC68030は例外ハンドラから戻って応答用CIRを読み出します。このようにして、メイン・プロセッサは応答用CIRを読み出し、受け取ったプリミティブを処理することにより、中断された命令の実行の継続を試みます。

10. 4. 20　命令実行後の例外処理要求プリミティブ

このプリミティブはコプロセッサから供給された例外ベクタ番号と“命令実行後の例外処理要求”スタック・フレーム・フォーマットを使用して例外処理を開始します。このプリミティブは、汎用または条件付きカテゴリの命令で使用できます。図10-44に“命令実行後の例外処理要求”プリミティブのフォーマットを示します。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	PC	0	1	1	1	0	1	ベクタ番号							

図10-42　“命令実行途中での例外処理要求”プリミティブのフォーマット

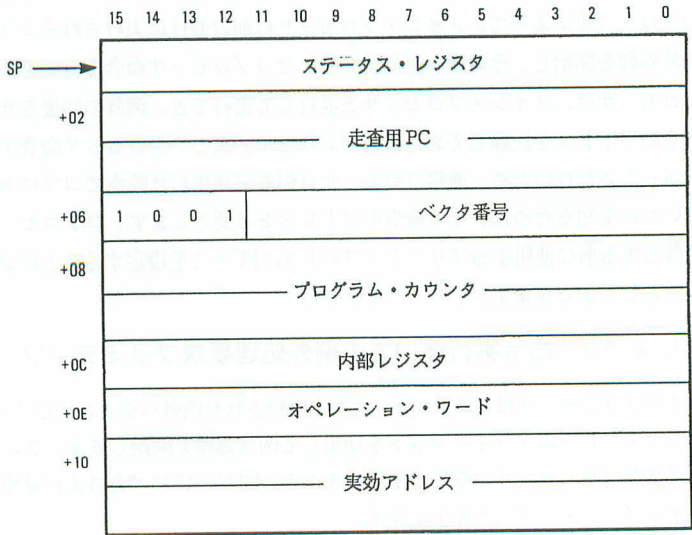



図10-43　MC68030の“命令実行途中での例外”スタック・フレーム

このプリミティブは上記の説明のとおりPCビットを使用します。ビット［07］には例外処理を開始するためにメイン・プロセッサが使用する例外ベクタ番号があります。

メイン・プロセッサがこのプリミティブを受け取ると、制御用CIRに例外アクノリッジ・マスク(「10. 3. 2　制御用CIR」参照)を書き込むことによって、コプロセッサの例外要求に認識応答します。

次に、MC68030は「8. 1　例外処理シーケンス」で説明する例外処理を実行します。この例外のベクタ番号は、このプリミティブのビット［0-7］から取り出され、MC68030は図10-45に示す6ワードのスタック・フレーム・フォーマットを使用します。

このプリミティブを受け取ったときのメイン・プロセッサの走査用PCの値は、“命令実行後の例外”スタック・フレームの走査用PCフィールドにセーブされます。セーブされるプログラム・カウンタの値は、このプリミティブを受け取ったときのコプロセッサ命令のF系列オペレーション・ワードのアドレスです。

MC68030は“命令実行後の処理要求”プリミティブを受け取ったとき、コプロセッサが命令を例外によって完了したか、アボートしたかのいずれかであるとみなします。例外ハンドラがスタック・フレームを変更しなかった場合、MC68030は例外ハンドラから戻り、スタック・フレームの走査用PCフィールドで指定されるロケーションから実行を開始します。このロケーションは次に実行する命令のアドレスのはずです。

コプロセッサはこのプリミティブを使用して、メイン・プロセッサが正規の応答を待っている間に命令を完了するかアボートしたときに、例外処理を要求します。汎用カテゴリの命令では応答は解放であり、条件付きカテゴリの命令では真/偽の条件判定結果です。したがって、このプリミティブに対するMC68030の応答動作は標準M68000ファミリの命令関連の例外処理(たとえば、0除算例外)に準じています。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	PC	0	1	1	1	1	0	ベクタ番号							

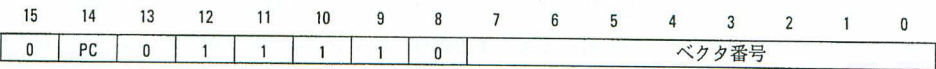



図10-44　“命令実行後の処理要求”プリミティブのフォーマット

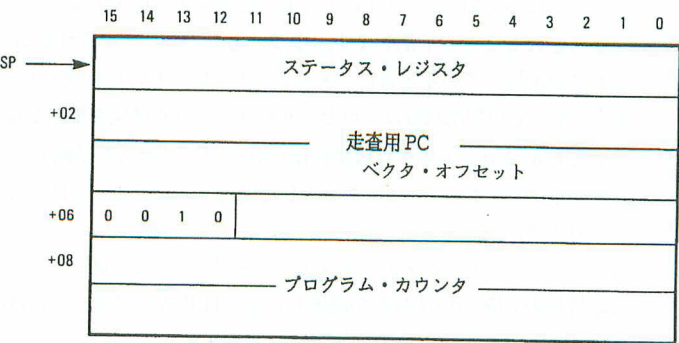



図10-45　MC68030の“命令実行後の例外”スタック・フレーム

10.5　例　外

コプロセッサ命令の実行に関連して、いくつかの例外状態が発生します。この例外をメイン・プロセッサが検出しようとコプロセッサが検出しようと、それに対する例外処理はメイン・プロセッサが調整し実行します。これらのコプロセッサ関連例外に対するサービスは、標準M68000ファミリの例外に対するサービスに使用するプロトコルの延長です。つまり、メイン・プロセッサが例外を検出するか、あるいはコプロセッサが例外の発生を通知すると、メイン・プロセッサは「8.1　例外処理シーケンス」で説明する例外処理に進みます。

10.5.1　コプロセッサが検出する例外

コプロセッサが検出する例外は、メイン・プロセッサが検出するものでもあり、通常コプロセッサ検出例外として分類されます。これらの例外は、M68000コプロセッサ・インタフェース動作、内部動作、またはコプロセッサの他のシステム関連動作中に発生する可能性があります。

大部分のコプロセッサ検出例外は、M68000コプロセッサ・インタフェースに定義されている3種類の“例外処理要求(Take Exception)”プリミティブの1つを使用してメイン・プロセッサに知らせます。メイン・プロセッサは上記の説明のとおり、これらのプリミティブに応答します。ただし、コプロセッサが検出したすべての例外が応答プリミティブによって通知されるわけではありません。cpSAVEまたはcpRESTORE命令の実行中にコプロセッサが検出したフォーマット・エラーは、「10.2.3.2.3　無効フォーマット・ワード」で説明した無効フォーマット・ワードを使用してメイン・プロセッサに通知されます。

10.5.1.1　コプロセッサ検出プロトコル違反

プロトコル違反例外は、M68000コプロセッサ・インタフェースを介して行なわれるメイン・プロセッサとコプロセッサ間の通信の障害です。コプロセッサ検出プロトコル違反は、メイン・プロセッサが予測しないシーケンスのコプロセッサ・インタフェース・レジスタ・セットのエントリにアクセスしたとき発生します。メイン・プロセッサが特定のコプロセッサ命令またはコプロセッサ応答プリミティブに対して実行する動作シーケンスについては、この章の前のほうで説明しています。

コプロセッサは、さまざまな方法でプロトコル違反を検出することができます。メイン・プロセッサは、M68000コプロセッサ・インタフェース・プロトコルに従って、常にコプロセッサの動作と同期して、オペレーション・ワード用、オペランド用、レジスタ選択用、命令アドレス用、あるいはオペランド・アドレス用の各CIRにアクセスします。つまり、メイン・プロセッサは特定のシーケンスでこれら5つのレジスタにアクセスし、コプロセッサはそのシーケンスでそれらのレジスタがアクセスされることを予期します。少なくとも、コプロセッサがコマンド用または条件用CIRへのアクセスを予期しているときに、メイン・プロセッサが上記の5つのレジスタのどれかにアクセスした場合、すべてのM68000コプロセッサがプロトコル違反を検出しなければなりません。同様に、コプロセッサがコマンド用または条件用CIRへのアクセスを予期していて、メイン・プロセッサがこれら5つのレジスタの1つにアクセスした場合、コプロセッサはプロトコル違反を検出し知らせる必要があります。

M68000コプロセッサ・インタフェース・プロトコルに従って、メイン・プロセッサはセーブ用または応答用のCIRのいずれかの読出し、またはリストア用あるいは制御用CIR に対する書込みを、コプロセッサ動作とは非同期に実行することができます。つまり、コプロセッサがある時点でのアクセスを明示的に予期しない状態で、これらのレジスタの1つに有効なアクセスを行なうことができます。コプロセッサはリストア用、応答用、および制御用のCIRに対するアクセスを予期することができますが、これらのレジスタはコプロセッサが予期しないときにもアクセスできるのです。

コプロセッサはcpSAVE命令またはcpRESTORE命令の実行中に、メイン・プロセッサにプロトコル違反を知らせることはできません。コプロセッサがcpSAVE命令またはcpRESTORE命令の実行中にプロトコル違反を検出した場合、次のコプロセッサ命令が開始されたときにその例外をメイン・プロセッサに知らせなければなりません。

コプロセッサ検出プロトコル違反の主な考え方は、コプロセッサは自分のインタフェース・レジスタの1つがアクセスされたとき、常に応答しなければならないということです。コプロセッサがそのアクセスを有効でないと判定した場合でも、コプロセッサとはメイン・プロセッサが次に応答用CIRを読み出す時点で、メイン・プロセッサに対して$\overline{\text{DSACKx}}$をアサートし、プロトコル違反を知らせる必要があります。コプロセッサが$\overline{\text{DSACKx}}$をアサートしなかった場合は、メイン・プロセッサはその信号がアサートされるのを(あるいは、他のバス終了信号を)、いつまでも待ち続けます。上記のプロトコルによって、コプロセッサがメイン・プロセッサをホルトしないことを保証しています。

コプロセッサは、“命令実行途中での例外処理要求”プリミティブによってメイン・プロセッサにプロトコル違反を知らせることができます。一貫性を維持するために、ベクタ番号はメイン・プロセッサ検出プロトコル違反に対するものと同じ13でなければなりません。メイン・プロセッサはこのプリミティブを読み出すと、「10. 4. 19　命令実行途中での例外処理要求プリミティブ」で説明する動作を実行します。例外ハンドラがスタック・フレームを変更しなかった場合、MC68030は例外ハンドラから復帰して応答用CIRを読み出します。

10. 5. 1. 2　コプロセッサ検出不当コマンドまたは条件ワード

不当コプロセッサ・コマンドまたは条件ワードは、コマンド用CIRまたは条件用CIRに書き込まれる値で、コプロセッサが認識しないものです。これらのレジスタのいずれかに書き込まれた値が正当でなかった場合、コプロセッサは“命令実行前の例外処理要求”プリミティブを応答用CIRに返さなければなりません。メイン・プロセッサがこのプリミティブを受け取ると、「10. 4. 18　命令実行前の例外処理要求プリミティブ」で説明する例外処理を実行します。例外ハンドラがメイン・プロセッサのスタック・フレームを変更しなかった場合、RTE命令によってMC68030は例外を実行した命令を再開します。コプロセッサを設計するときは、システムが認識されないコマンドまたは条件ワードのエミュレーションをサポートしている場合は、コプロセッサの状態が不当コマンドまたは条件例外によって回復不可能なほど変更されないように保証しなければなりません。

モトローラのM68000コプロセッサはすべて、F系列エミュレータ例外ベクタ番号11をもつ“命令実行前の例外処理要求”プリミティブを返すことによって、不当コマンドおよび条件ワードを知らせます。

10. 5. 1. 3　コプロセッサ・データ処理例外

コプロセッサの内部動作に関連する例外は、データ処理関連例外として分類されています。これらの例外は、M68000マイクロプロセッサで定義されている“0による除算”例外に類似しており、該当する例外ベクタ番号を含む3種類の“例外処理要求”プリミティブの1つを使用して、メイン・プロセッサに知らせる必要があります。例外を知らせるために、この3種類のプリミティブのうちどれを使用するかは、一般に例外処理終了後にメイン・プロセッサが命令動作のどのポイントからプログラム・フローを継続するかによって決まります。これについては、「10. 4. 18　命令実行前の例外処理要求プリミティブ」、「10. 4. 19　命令実行途中での例外処理要求プリミティブ」、および「10. 4. 20　命令実行後の例外処理要求プリミティブ」を参照してください。

10. 5. 1. 4　コプロセッサ・システム関連の例外

DMAコプロセッサで検出されるシステム関連例外には、バス動作に関連するものとコプロセッサの外部で発生するその他の例外(たとえば、割込み)があります。コプロセッサおよびメイン・プロセッサが実行する動作は、発生する例外の種類によって異なります。

DMAコプロセッサがアドレス・エラーまたはバス・エラーを検出したら、コプロセッサはシステムでアクセス可能なレジスタに、メイン・プロセッサの例外処理ルーチンが必要とする情報を格納する必要があります。コプロセッサは、該当する例外ベクタの番号をもつ3種類の“例外処理要求”プリミティブの1つを応答用CIRに入れなければなりません。3種類のプリミティブのうちどれを使用するかは、その例外が検出されたコプロセッサ命令でのポイント、および例外処理終了後にメイン・プロセッサがプログラムの実行を継続する命令でのポイントによって異なります。

10. 5. 1. 5　フォーマット・エラー

応答プリミティブを使用してメイン・プロセッサに通知されないコプロセッサ検出例外は、フォーマット・エラーだけです。cpRESTORE命令の実行中にメイン・プロセッサがフォーマット・ワードをリストア用CIRに書き込むと、コプロセッサはそのワードをデコードして、それが有効であるかどうかを判断します(「10. 2. 3. 3　コプロセッサ・コンテキスト・セーブ命令」参照)。そのフォーマット・ワードが有効でなかった場合、コプロセッサは無効フォーマット・コードをリストア用CIRに入れます。メイン・プロセッサが、無効フォーマット・コードを読み出すと、アボート・マスク(「10. 3. 2　制御用CIR 」参照)を制御用CIRに書き込むことによって、そのコプロセッサ命令をアボートします。次に、メイン・プロセッサは、4ワードの“命令実行前の例外”スタック・フレームとフォーマット・エラーの例外ベクタ番号14を使用して例外処理を実行します。したがって、例外ハンドラがスタック・フレームの内容を変更しなかった場合、MC68030は例外ハンドラでRTE 命令が実行されると、cpRESTORE命令を再開します。メイン・プロセッサがセーブ用CIRを読み出してcpSAVE命令を開始したときに、コプロセッサが無効フォーマット・コードを返した場合、メイン・プロセッサは上記のcpRESTORE命令の場合と同様に、フォーマット・エラー例外処理を実行します。

10. 5. 2　メイン・プロセッサ検出例外

コプロセッサ命令の実行に関連する例外の多くは、コプロセッサではなくメイン・プロセッサが検出します(メイン・プロセッサがサービスする)。これらの例外は、コプロセッサ応答プリミティブの実行、M68000コプロセッサ・インタフェースを介しての通信、またはメイン・プロセッサによる条件付きコプロセッサ命令の終了に関連していることがあります。

10. 5. 2. 1　プロトコル違反

メイン・プロセッサは、応答用プリミティブから有効でないプリミティブを読み出したとき、プロトコル違反を検出します。M68000コプロセッサ・インタフェースに定義されているプリミティブに応答して発生する可能性のあるプロトコル違反を、表10-6にまとめて示します。

MC68030はプロトコル違反を検出しても、自動的に結果の例外処理を制御用CIRに書き込んで、コプロセッサに知らせることはありません。しかし、例外処理ルーチンはMOVES命令を使用して応答用CIRから読出しを行ない、MC68030がプロトコル違反例外処理を開始する原因となったプリミティブを確認します。メイン・プロセッサは“命令実行途中での例外”スタック・フレーム(図10-43参照)およびプロトコル違反例外ベクタ番号13を使用して例外処理を開始します。例外ハンドラがスタック・フレームを変更しなかった場合、メイン・プロセッサはRTE命令を実行して例外ハン

ドラから戻った後、再び応答用CIRを読み出します。このプロトコルによって、ハードウェアでサポートされていないM68000コプロセッサ・インタフェースの拡張を、メイン・プロセッサのソフトウェアでエミュレートすることができます。したがって、プリミティブの実行をメイン・プロセッサがソフトウェアでエミュレートできれば、コプロセッサはプロトコル違反に気づきません。

10. 5. 2. 2　F系列エミュレータ例外

MC68030が検出するF系列エミュレータ例外は、明示的にあるいは暗黙に命令ストリームのF系列オペレーション・ワードのエンコーディングに関連しています。メイン・プロセッサが、F系列の

表10-6　プリミティブ処理関連の例外

プリミティブ	プロトコル	F系列	その他
ビジー			
ヌル			
スーパバイザ・チェック* その他：“S” ビット＝0であれば特権違反			X
オペレーション・ワードの転送*			
命令ストリームからの転送* プロトコル：長さフィールドが奇数の場合(長さ0は正当)	X		
実効アドレスの評価および転送 プロトコル：条件付き命令で使用した場合 F系列：オペレーション・ワードのEAが制御可変でなかった場合	X	X	
実効アドレスの評価およびデータの転送 プロトコル： 1. 条件付き命令で使用した場合 2. 長さが1、2、または4 以外で、EA＝レジスタ直接の場合 3. EA＝イミディエイト、長さが奇数で、1より大きい場合 4. 非可変のアドレスへ書込みを行なおうとした場合(プリミティブのアドレスが正当であっても) F系列：有効EAフィールドがオペレーション・ワードのEAと一致していない場合	X	X	
評価済み実効アドレスへの書込み プロトコル：条件付き命令で使用した場合	X		
ビジー			
アドレス取得およびデータ転送*			
スタック先頭との転送* プロトコル：長さフィールド値が1、2、または4 以外	X		
メイン・プロセッサのレジスタとの転送*			
メイン・プロセッサの制御レジスタとの転送 プロトコル：無効制御レジスタ選択コード	X		
複数のメイン・プロセッサ・レジスタの転送*			
複数のコプロセッサ・レジスタの転送 プロトコル： 1. 条件付き命令で使用した場合 2. 長さが奇数の場合 F系列： 1. EAが制御可変でない、あるいはCPからメモリへの転送で(An)＋を指定した場合 2. EAが制御可変でない、あるいはメモリからCPへの転送で－(An)を指定した場合	X X		
ステータスおよび走査用PC、あるいはそのいずれかの転送 プロトコル：条件付き命令で使用した場合 その他： 1. トレース――MC68020が“フローの変化でのトレース”モードになっていてDR＝1の場合はトレース・モードの保留 2. アドレス・エラー――走査用PCに奇数値が書き込まれた場合	X		X
命令実行前、命令実行途中、または命令実行後の例外 例外の種類はプリミティブで供給されるベクタによる	X	X	X

*CA＝0 でこのプリミティブを使用した場合、条件付き命令でプロトコル違反が発生します。
略語：
EA＝実効アドレス
CP＝コプロセッサ

オペレーション・ワードが有効でないと判断すると、F系列エミュレータ例外処理を開始します。ビット［8：6］＝110または111のF系列オペレーション・ワードによって、MC68030はその命令に対してコプロセッサと交信を行なうことなく例外処理を開始します。また、命令セット(「第3章 命令セット」参照)の中で有効なコプロセッサ命令の1つにマップされない、ビット［8：6］＝000101をもつオペレーション・ワードによって、MC68030はF系列エミュレータ例外処理を開始します。上記の2つの状況のいずれかによってF系列エミュレータ例外処理が発生した場合、メイン・プロセッサは例外処理を開始する前に制御用CIRへの書込みは行ないません。

F系列例外は、コプロセッサ応答プリミティブが要求する操作が、コプロセッサ命令のオペレーション・ワードのビット［0-5］の実効アドレスに適合しない場合にも発生します。M68000コプロセッサ応答プリミティブを使用したときに発生する可能性のあるF系列エミュレータ例外を、表10-6に要約します。例外が無効プリミティブを受け取ることによって発生した場合、メイン・プロセッサはF系列エミュレータ例外処理を開始する前に、制御用CIRにアボート・マスクを書き込むことによって、実行中のコプロセッサ命令をアボートします。

コプロセッサ命令を開始したコプロセッサ・インタフェース・レジスタ・アクセス中にバス・エラーが発生すると、別のタイプのF系列エミュレータ例外が発生します。メイン・プロセッサはコプロセッサが存在しないものとみなして、例外処理を実行します。

メイン・プロセッサはF系列エミュレータ例外処理を開始するときは、4ワードの“命令実行前の例外”用スタック・フレーム(図10-41参照)およびF系列エミュレータ例外ベクタ番号11を使用します。したがって、例外ハンドラがスタック・フレームを変更しなかった場合、メイン・プロセッサはRTE 命令を実行して例外ハンドラから戻った後、その例外を発生した命令の再開を試みます。

F系列例外の原因をソフトウェアでエミュレートできる場合、ハンドラはエミュレーションの結果をプログラミング・モデルにある適当なレジスタ、およびセーブされたスタック・フレームのステータス・フィールドの中に格納します。例外ハンドラは、セーブされたスタック・フレームのプログラム・カウンタ・フィールドが次の命令のオペレーション・ワードを指すように調整してからRTE命令を実行します。次に、MC68030はエミュレートされた命令の次の命令を実行します。

例外ハンドラはスタックにあるステータス・レジスタのコピーをチェックして、トレーシングがオンになっているかどうか確認しなければなりません。トレーシングがオンになっている場合は、トレース例外処理もエミュレートする必要があります。詳細については、「8. 1. 7　トレース例外」を参照してください。

10. 5. 2. 3　特権違反

特権違反はcpSAVE命令およびcpRESTORE命令、そしてスーパバイザ・チェック・コプロセッサ応答プリミティブによっても発生する可能性があります。メイン・プロセッサがユーザ状態(ステータス・レジスタのS＝0)にあるときに、cpSAVE命令またはcpRESTORE命令のいずれかを実行しようとすると、メイン・プロセッサは特権違反例外処理を開始します。メイン・プロセッサは、cpSAVE命令またはcpRESTORE命令に関係するコプロセッサと交信を行なう前に、この例外処理を開始します。

メイン・プロセッサが、スーパバイザ・チェック・プリミティブを読み出したときに、ユーザ状態でコプロセッサ命令を実行していた場合、メイン・プロセッサは制御用CIRにアボート・マスク(「10. 3. 2　制御用CIR」参照)を書き込むことによって、実行中のコプロセッサ命令をアボートします。その後、メイン・プロセッサは特権違反例外処理を開始します。

特権違反が発生した場合、メイン・プロセッサは4ワードの“命令実行前の例外”用スタック・フレーム(図10-41参照)および特権違反例外ベクタ番号8を使用して例外処理を開始します。したがって、例外ハンドラがスタック・フレームを変更しなかった場合、メイン・プロセッサはRTE命令

を実行してハンドラから戻った後、その例外を発生した命令の再開を試みます。

10. 5. 2. 4　cpTRAPcc命令トラップ

cpTRAPcc命令の実行中に、コプロセッサがヌル・プリミティブを使用してメイン・プロセッサに「真」の条件判定結果を返した場合、メイン・プロセッサはトラップ例外処理を開始します。メイン・プロセッサは、6ワードの“命令実行後の例外”スタック・フレーム(図10-45参照)およびトラップ例外ベクタ番号7を使用します。このスタック・フレームの走査用PCフィールドには、cpTRAPcc命令の次の命令のアドレスが含まれています。次に、cpTRAPcc命令に関連する処理を開始でき、例外ハンドラは6ワードのスタック・フレームに含まれている情報を使用して、cpTRAPcc命令にエンコードされたイミディエイト・オペランド・ワードを見つけることができます。例外ハンドラがスタック・フレームを変更しなかった場合、メイン・プロセッサはRTE命令を実行してハンドラから抜け出した後、cpTRAPcc命令の次の命令を実行します。

10. 5. 2. 5　トレース例外

MC68030ccは、「8. 1. 7　トレース例外」で説明した2つの命令トレース・モードをサポートします。“命令実行ごとのトレース”モードでは、MC68030は各命令の終了後にトレース例外を処理します。“命令フローの変化時でのトレース”モードでは、MC68030はステータス・レジスタを変更する命令、あるいはプログラム・カウンタに次の命令以外のアドレスを置く命令を実行するたびにトレース例外を処理します。

コプロセッサcpSAVE命令、cpRESTore命令、または条件付きカテゴリの命令を実行するために使用するプロトコルは、メイン・プロセッサでトレース例外が保留されているときにも変わりません。メイン・プロセッサはその命令の実行終了後に、保留されている“命令実行ごとのトレース”例外を実行します。“命令フローの変化時でのトレース”モードになっていて、命令が次の命令以外のアドレスをプログラム・カウンタに置いた場合、メイン・プロセッサはその命令を実行した後トレース例外処理を実行します。

汎用カテゴリの命令の実行中にトレース例外が保留されていない場合、メイン・プロセッサはCA＝0のプリミティブを読み出したあと、コプロセッサとの交信を終了します。したがって、コプロセッサはメイン・プロセッサの命令実行と並行してcpGEN命令を完了することができます。しかし、トレース例外が保留されているときは、メイン・プロセッサはトレース例外を処理する前に、cpGEN命令に関するすべての処理が終了したことを確認しなければなりません。この場合、メイン・プロセッサはCA＝0、PF＝1のヌル・プリミティブを受け取るか、“命令実行後の例外処理要求”プリミティブによって開始された例外処理が終了するまで、応答用CIRを継続に読み出し、そのプリミティブに対するサービスを実行します。コプロセッサはcpGEN命令の実行中に、PF＝0となっているCA＝0ヌル・プリミティブを返さなければなりません。メイン・プロセッサは、これらのプリミティブでIA＝1となっている場合は、応答用CIRを読み出す合間に、保留されている割込みをサービスすることができます(表10-3参照)。このプロトコルは、cpGEN命令に関係するすべての処理が終了するまで、トレース例外の処理が開始されないよう保証しています。

汎用カテゴリの命令が開始された時点で、MC68030のステータス・レジスタでT1/T0＝01(“フロー変化でのトレース”モード)となっていた場合、その命令の実行中にコプロセッサがDR＝1で“ステータス・レジスタおよび走査用PCの転送”プリミティブを発行したときだけ、その命令に対するトレース例外処理が行なわれます。この場合、メイン・プロセッサがトレース例外ハンドラの実行を開始したときに、コプロセッサがcpGEN命令をそのまま並行して実行している可能性があります。したがって、“フローの変化でのトレース”例外ハンドラの中で実行されたcpSAVE命令が、並行動作中のcpGEN命令の実行を中断することもあります。

10. 5. 2. 6　割込み

「8. 1. 9　割込み例外」で説明した割込み処理は、任意の命令境界で発生することができます。また、次の2つの状態のいずれかにおいて、汎用または条件付きカテゴリの命令の実行中に割込みがサービスされることもあります。

メイン・プロセッサがCA＝1、IA＝1のヌル・プリミティブを読んだ場合、応答用CIRを読み出す前に、保留されている割込みがあれば、それに対するサービスを実行します。同様に、cpGEN命令の実行中にトレース例外が保留されていて、メイン・プロセッサがCA＝0、IA＝1、PF＝0のヌル・プリミティブを読み出した場合(「10. 5. 2. 5　トレース例外」参照)、メイン・プロセッサは再び応答用CIRの読出しを行なう前に、保留されている割込みのサービスを実行します。

MC68030は、汎用または条件付きカテゴリの命令の実行中に割込みをサービスするときは、10ワードの“命令実行途中での例外”スタック・フレームを使用します。スタック・フレームを使用することによって、メイン・プロセッサは必要なすべての処理を実行してから応答用CIRの読出しに戻ります。これによって、その割込み例外を受け付けたときに実行中であったコプロセッサ命令を継続して実行することができます。

MC68030は、cpSAVE命令の実行中にセーブ用CIRから“ノット・レディ”フォーマット・ワードを読み出した場合は、割込みのサービスも実行します。MC68030は“ノット・レディ”フォーマット・ワードを読み出したあと、割込みのサービスを行なうときは、通常4ワードの“命令実行前の例外”スタック・フレームを使用します。このようにして、プロセッサは保留されている割込みをサービスし、RTE命令を実行してハンドラから戻り、セーブ用CIRを読み出すことによってcpSAVE命令を再開することができます。

10. 5. 2. 7　メイン・プロセッサ検出フォーマット・エラー

MC68030はcpSAVE命令またはcpRESTORE命令の実行中に、有効なフォーマット・ワードの長さフィールドの値が4バイトの倍数でなかった場合は、フォーマット・エラーを検出することができます。MC68030がcpSAVE命令の実行中に、セーブ用CIRから無効な長さフィールドのフォーマット・ワードを読み出した場合は、制御用CIRにアボート・マスク(「10. 3. 2　制御用CIR」参照)を書き込むことによって、そのコプロセッサ命令をアボートし、フォーマット・エラー例外処理を開始します。MC68030がcpRESTORE命令で指定された実効アドレスから、無効な長さフィールドをもつフォーマット・ワードを読み出した場合、MC68030はそのフォーマット・ワードをコプロセッサのリスト用CIRに書き込み、ついでリストア用CIRからコプロセッサの応答を読み出します。次に、MC68030は制御用CIRにアボート・マスク(「10. 3. 2　制御用CIR」参照)を書き込むことによってcpRESTORE命令をアボートし、フォーマット・エラーの例外処理を開始します。

MC68030は、フォーマット・エラーの例外処理を開始するとき、4ワードの“命令実行前の例外”スタック・フレームとフォーマット・エラー・ベクタ番号14を使用します。したがって、例外ハンドラがスタック・フレームを変更しなかった場合、メイン・プロセッサはRTE命令を実行して例外ハンドラから抜け出した後、その例外を発生した命令の再開を試みます。

10. 5. 2. 8　アドレス・エラーおよびバス・エラー

コプロセッサ命令関連のバス・フォールトは、コプロセッサと交信するためのCPU空間へのメイン・プロセッサ・バス・サイクル、あるいはコプロセッサ命令実行の一部であるメモリ・サイクルで発生する可能性があります。コプロセッサ命令を開始するのに使用するコプロセッサ・インタフェース・レジスタへのアクセス時に、バス・エラーが発生した場合、メイン・プロセッサはシステムにコプロセッサが存在しないものとみなし、F系列エミュレータ例外(「10. 5. 2. 2　F系列エミュ

レータ例外」参照)処理を実行します。つまり、コプロセッサ命令によるCIRへの最初のアクセスでエラーが発生した場合、プロセッサはF系列エミュレータ例外処理を実行します。その他のコプロセッサ・アクセス、またはコプロセッサ命令の実行中のメモリ・アクセスにおいてバス・エラーが発生した場合、メイン・プロセッサはバス・エラーの例外処理(「8. 1. 2　バス・エラー例外」参照)を実行します。例外ハンドラがバス・エラーの原因を是正した後、メイン・プロセッサはコプロセッサ命令でその障害が発生したところまで戻ることができます。

アドレス・エラーは、MC68030が奇数アドレスから命令をフェッチしようとした場合に発生します。これはcpBcc命令またはcpDBcc命令のデスティネーション・アドレスの計算値が奇数であった場合、あるいは“ステータス・レジスタおよび走査用PCの転送”プリミティブによって走査用PCに奇数アドレスが転送された場合に発生します。アドレス・エラーが発生した場合、MC68030は「8. 1. 3　アドレス・エラー例外」で説明した例外処理を実行します。

10. 5. 3　コプロセッサのリセット

外部リセット信号またはRESET命令のいずれかでシステムの外部デバイスをリセットすることができます。システム設計者は、両タイプのリセットを使用して、あるいは外部リセット信号だけで、コプロセッサをリセットし初期化するよう設計することができます。MC68030の設計との一貫性を維持するために、コプロセッサは外部システム・リセットによってのみ影響され、RESET命令の影響を受けないようにしておくべきです。その理由は、コプロセッサはメイン・プロセッサのプログラミング・モデルおよびMC68030の内部状態の拡張と考えられるためです。

10. 6　コプロセッサ命令の要約

コプロセッサ命令のフォーマットは、「3. 9　命令フォーマットの要約」に含まれています。

以下にコプロセッサ応答プリミティブのフォーマットの要約を示します。ビット [13 : 8] = $ 00または$ 3Fの応答プリミティブは、常にプロトコル違反を発生します。ビット [13 : 8] = $ 0B、$ 18-$ 1B、$ 1F、$ 28-$ 2B、および$ 38-$ 3Bの応答プリミティブは、現在はプロトコル違反を発生しますが、これらは未定義となっており将来の使用に備えてモトローラが予約しています。

ビジー

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	PC	1	0	0	1	0	0	0	0	0	0	0	0	0	0

複数のコプロセッサ・レジスタの転送

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	DR	0	0	0	0	1	長さ							

ステータス・レジスタおよび走査用PCの転送

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	DR	0	0	0	1	SP	0	0	0	0	0	0	0	0

スーパバイザ・チェック

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	PC	0	0	0	1	0	0	0	0	0	0	0	0	0	0

アドレス取得およびデータ転送

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	DR	0	0	1	0	1	長さ							

複数のメイン・プロセッサ・レジスタの転送

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	DR	0	0	1	1	0	0	0	0	0	0	0	0	0

オペレーション・ワード転送

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	0	0	0	1	1	1	0	0	0	0	0	0	0	0

ヌル

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	0	0	1	0	0	IA	0	0	0	0	0	0	PF	TF

実効アドレスの評価および転送

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	0	0	1	0	1	0	0	0	0	0	0	0	0	0

単独メイン・プロセッサ・レジスタの転送

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	DR	0	1	1	0	0	0	0	0	0	D/A	レジスタ		

メイン・プロセッサの制御レジスタの転送

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	DR	0	1	1	0	1	0	0	0	0	0	0	0	0

スタックの先頭との転送

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	DR	0	1	1	1	0	長さ							

命令ストリームからの転送

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	0	0	1	1	1	1	長さ							

実効アドレス評価およびデータ転送

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	DR	1	0	有効EA			長さ							

命令実行前の例外処理要求

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	PC	0	1	1	1	0	0	ベクタ番号							

命令実行途中での例外処理要求

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	PC	0	1	1	1	0	1	ベクタ番号							

命令実行後の例外処理要求

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	PC	0	1	1	1	1	0	ベクタ番号							

評価済み実効アドレスへの書込み

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	1	0	0	0	0	0	長さ							

第 11 章

命令実行時間

本章では、MC68030の命令実行および操作(テーブル・サーチなど)時間を外部クロック・サイクル数の単位で示します。ここで述べる情報は的確な実行および操作の実行時間のガイドラインにはなりますが、あらゆる動作環境における正確な実行時間を示すものではありません。サイクル数で実行時間を表わしているのは、命令または操作の正確な実行時間はメモリ速度と他の変数に大きく依存するためです。本章で示す実行時間によって、アセンブラやコンパイラのプログラマは、MC68030の性能を評価するのに必要な実際のキャッシュ・ケースおよび平均ノー・キャッシュ・ケースの実行時間を予測することができます。加えて、例外処理、コンテクストの切換え、および割込み処理の実行時間も含まれているため、マルチタスキングまたはリアルタイム・システムの設計者は、タスク切換えのオーバヘッド、最大割込み待ち時間、および同様な実行時間のパラメータを予測することができます。

クロック周波数への依存を避けるため、本章では命令および操作時間をクロック・サイクルで示しています。

11.1　性能トレード・オフ

MC68030は最悪時の性能を犠牲にして、平均的な性能を最大限に高めています。1命令の実行に要する時間は、0から100クロック以上までさまざまです。実行時間に影響を与える要素は、前後の命令、命令ストリームのアラインメント、キャッシュでのオペランドおよび命令の滞在性、アドレス変換キャッシュでのアドレス変換値の滞在性、およびオペランドのアラインメントです。

MC68030の平均性能を向上させるために、ベスト・ケースの性能を上げ、ワースト・ケースの動作が発生する機会を減らすために一定のトレード・オフを設けています。たとえば、バースト充てんは、後でアクセスするためのデータをプリフェッチすることにより性能を高めていますが、そのため外部バス・コントローラやキャッシュにより長い時間をゆだねています。

MC68030はデータの書込みを、命令キャッシュの読出し、データ・キャッシュの読出し、およびマイクロシーケンサの実行、あるいはそのいずれかとオーバラップすることができます。命令キャッシュの読出しを、データ・キャッシュの充てんおよびマイクロシーケンサ・アクティビティ、またはそのいずれかとオーバラップさせることができます。同様に、データ・キャッシュの読出しを、命令キャッシュの充てん、マイクロシーケンサ・アクティビティ、またはそのいずれかとオーバラップすることができます。オンチップ・レジスタだけにアクセスする命令の実行は、前の命令が実行したプリフェッチが命令キャッシュの中にあれば、その命令が実行した並行データ書込みと全体的にオーバラップすることができます。

11．2　資源のスケジューリング

命令実行時間の可変性のいくつかは、資源利用のオーバラップによるものです。プロセッサは、8つの独立してスケジュールされる資源で構成されているとみなすことができます。スケジューリングが命令の境界に直接関係することはほとんどないので、命令を実行している完全なコンテキストを知らないで、特定の命令を実行するのに必要な時間を正確に予測することは不可能です。MC68030内部のこれらの資源の位置を図11-1に示します。

11．2．1　マイクロシーケンサ

マイクロシーケンサはマイクロ命令を実行しているか、あるいはマイクロコードを継続して実行するのに必要なアクセスが完了するのを待っているかのいずれかです。バス・コントローラはすべてのバス・アクティビティに責任をもっています。マイクロシーケンサは、バス・コントローラ、命令の実行、そして実効アドレスの計算やコンディション・コードのセットなどの内部プロセッサの動作を制御します。マイクロシーケンサは命令ワードのプリフェッチを開始し、命令パイプ内の命令ワードの妥当性を制御します。

11．2．2　命令パイプ

MC68030は3ワードの命令パイプを内蔵し、それによって命令オペコードをデコードします。図11-1に示すように、命令ワード(命令オペレーション・ワードおよびすべての拡張ワード)は、ステージBでパイプに入り、ステージCおよびDに進みます。命令ワードはパイプのステージDに達すると完全にデコードされます。パイプの各ステージには、ステージのワードに異常終了したバス・サイクルからデータがロードされたか否かを反映するステータス・ビットがあります。パイプのステージはマイクロシーケンサが発行した特別なプリフェッチ要求への応答によってのみ充てんされます。

ワードはキャッシュ保持レジスタから命令パイプにロードされます。パイプの個々のステージは16ビット幅しかありませんが、キャッシュ保持レジスタは32ビット幅で、完全なロング・ワードが入ります。このロング・ワードはマイクロシーケンサからのプリフェッチ要求に応答して、命令キャッシュまたは外部バスから得られます。マイクロシーケンサが偶数ワード(ロング・ワードに整列)プリフェッチを要求すると、命令キャッシュまたは外部バスから完全なロング・ワードがアクセスされ、キャッシュ保持レジスタにロードされます。また、上位ワードもパイプのステージBにロードされます。その後、次のシーケンシャル・プリフェッチの命令ワードをキャッシュ保持レジスタから直接アクセスすることができ、外部バス・サイクルや命令キャッシュ・アクセスは必要ありません。キャッシュ保持レジスタは、命令キャッシュがイネーブルされているかディセーブルされているかに関係なく、パイプに命令ワードを供給します。

プリフェッチ要求はキャッシュ保持レジスタ、命令キャッシュ、およびバス・コントローラに同時に提供されます。そのため、命令キャッシュがディセーブルされた場合は、命令プリフェッチがキャッシュ保持レジスタでヒットし、外部バス・サイクルをアボートします。

11．2．3　命令キャッシュ

命令キャッシュは、マイクロシーケンサの命令プリフェッチ部分をサービスします。オンチップ命令キャッシュでヒットした命令のプリフェッチは、そのプリフェッチに外部バス・アクティビティが不要のため、命令の実行に遅延が生じることはありません。また、命令キャッシュは、命令キャッシュ・ミスの後の命令キャッシュの充てん中に外部バスとやりとりを行ないます。

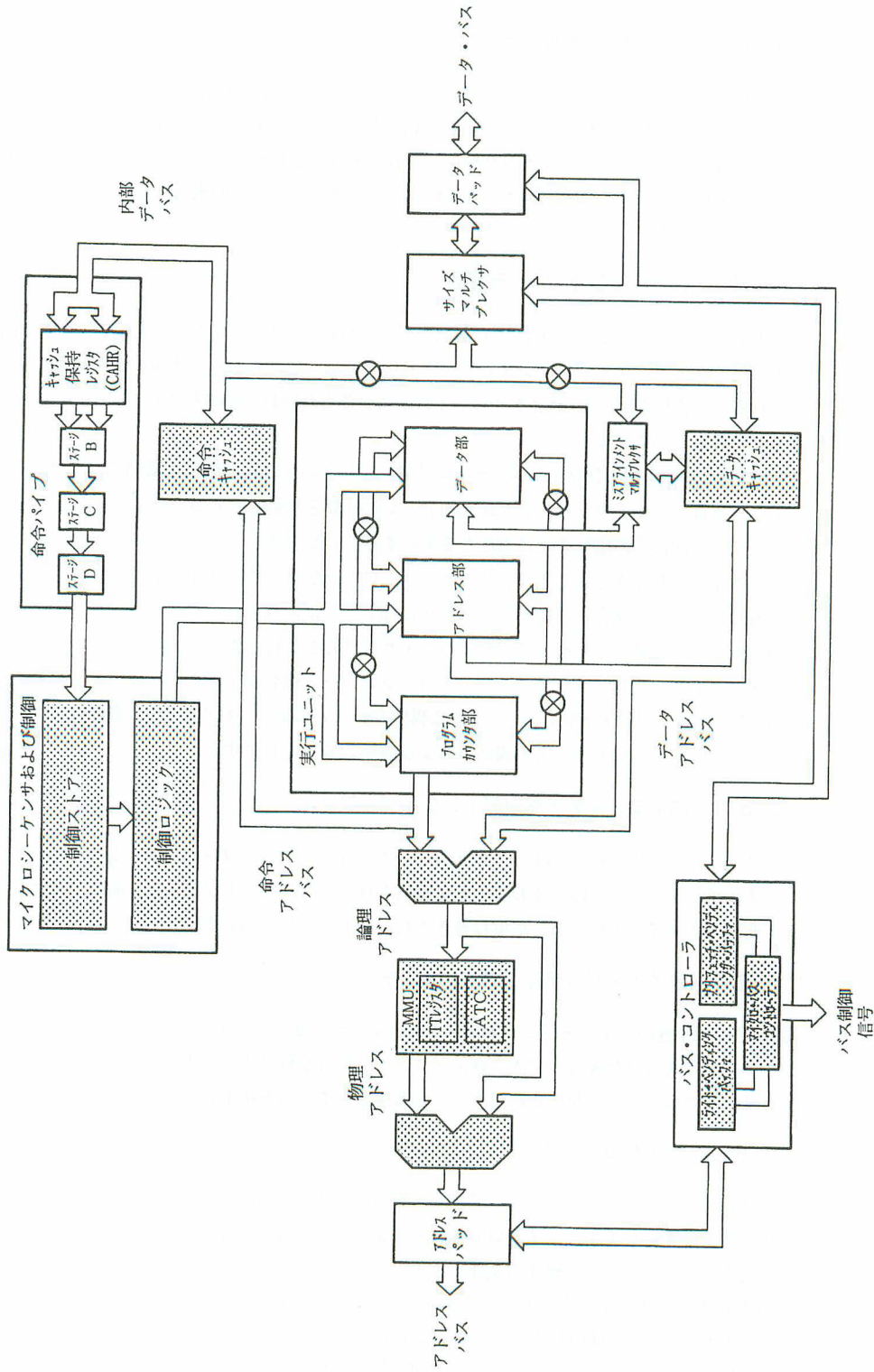



図11-1　MC68030ブロック図 ―― 8つの独立した資源

11. 2. 4　データ・キャッシュ

データ・キャッシュはデータの読出しをサービスし、データの書込み時に更新されます。データ・キャッシュからアクセスされる実行ユニットが要求するデータ・オペランドは、データ・フェッチのための外部バス・アクティビティのために、命令の実行が遅れることはありません。データ・キャッシュもまたデータ・キャッシュ・ミスに続くデータ・キャッシュの充てん中に、外部バスとやりとりを行ないます。

11. 2. 5　バス・コントローラ資源

命令キャッシュでミスしたプリフェッチによって、外部メモリ・サイクルが実行されます。同様に、オンチップ・データ・キャッシュでデータ・リード・ミスがあると、外部メモリ・サイクルが要求されます。これらのバス・サイクルのいずれかに必要な時間は、他の内部アクティビティとオーバラップすることができます。

バス・コントローラおよびマイクロシーケンサは、1つの命令に対して同時に動作できます。バス・コントローラは、マイクロシーケンサが実効アドレス計算を制御したりコンディション・コードをセットしている間に、読出しまたは書込みを実行することができます。また、マイクロシーケンサがバス・サイクルを要求したときに、バス・コントローラがそのサイクルをすぐには実行できないこともあります。その場合、このバス・サイクルはキューに入れられ、バス・コントローラは現在実行中のサイクルを終了した時点で、そのバス・サイクルを実行します。

バス・コントローラは、マイクロ・バス・コントローラ、命令フェッチ・ペンディング・バッファ、およびライト・ペンディング・バッファで構成されています。これらの3つの資源は、オンチップ・キャッシュでミスしたすべての書込みおよび読出しを実行します。

11. 2. 5. 1　命令フェッチ・ペンディング・バッファ

命令フェッチ・メカニズムには、1個のロング・ワード命令フェッチ・ペンディング・バッファがあります。インタロックを備えており、前に要求されたプリフェッチが完了する前に、命令プリフェッチ要求によってこのバッファが重ね書きされないようにしています。

11. 2. 5. 2　ライト・ペンディング・バッファ

MC68030は1個のライト・ペンディング・バッファを内蔵し、ライト・サイクルの要求がバス・コントローラに送られた後も、マイクロシーケンサが実行を継続できるようにしています。インタロックはマイクロシーケンサがこのバッファを重ね書きするのを防止します。

11. 2. 5. 3　マイクロ・バス・コントローラ

マイクロ・バス・コントローラは、プロセッサの他の部分からバス・コントローラに出されたバス・サイクルを実行します。マイクロ・バス・コントローラは必要なダイナミック・バス・サイジングを実現し、またバースト操作も制御します。

外部メモリからの命令をプリフェッチするとき、マイクロ・バス・コントローラはロング・ワードのリード・サイクルを使用します。プロセッサは2ワードを読み出しますので、一度に2つの命令または複数ワード命令の2ワードをキャッシュ保持レジスタ(イネーブルされかつ凍結されていない場合は命令キャッシュにも)にロードすることができます。命令のプリフェッチが奇数ワード境界で行なわれる場合に、ある命令ワードにプリフェッチを行なったときに、命令キャッシュ・ミスにより、それがキャッシュ保持レジスタで見つからなかったとき(たとえば、奇数ワード・ロケーションへの分岐)に、特別なケースが発生します。MC68030は32ビット・メモリから、ロング・ワードの

ベース・アドレスに関連する偶数および奇数ワードの両方を1バス・サイクルで読み出します。また、8ビットまたは16ビット・メモリからは、奇数ワードの前に偶数ワードを読み出します。この偶数および奇数ワードは両方ともキャッシュ保持レジスタ(および命令キャッシュ――イネーブルされかつ凍結されていない場合)にロードされます。

11.2.6　メモリ管理ユニット

MC68030は必要なときに外部アクセスのために論理アドレスを物理アドレスに変換するメモリ管理ユニット(MMU)を内蔵します。

MMUはアドレス変換キャッシュ(ATC)を用いて最も近い時点で使用された変換値を記憶します。論理アドレスに対応する物理アドレスがATCにあれば、アドレス変換時間はオンチップ・キャッシュ・アクセスと完全にオーバラップし、命令の実行時間には影響を与えません。

ATCがある論理アドレスに対する変換値をもっていないときには、プロセッサは外部メモリに対するテーブル・サーチ操作を実行します。テーブル・サーチに要する時間はアドレス変換ツリーの構造および変換ツリーの非常駐部分が必要か否かによって異なります。

MMUはデマンド・ページ方式の仮想メモリをサポートします。テーブル・サーチが例外の発生で終了し、要求された命令またはデータが存在しないことを示すと、適当なページをメモリに入れるための余分な時間が必要です。この所要時間はその例外に対する処理ルーチンに依存します。

11.3　命令実行時間の計算

命令-キャッシュ・ケースの実行時間、オーバラップ、平均・ノー・キャッシュ・ケースの実行時間、および実際の命令－キャッシュ・ケースの実行時間の計算を以下の各項で説明します。

11.3.1　命令-キャッシュ・ケース

ある命令に対する命令-キャッシュ・ケース時間(CC)は、対応するすべての命令プリフェッチがオンチップ命令キャッシュに存在する場合に、その命令を実行するのに必要な合計クロック周期数です。すべてのバス・サイクルは2クロック周期と仮定しています。命令キャッシュ・ケース時間は、他の命令とオーバラップしないものと仮定し、またオンチップ・データ・キャッシュでのヒットを考慮に入れていません。一部の命令に対する全体的な命令キャッシュ・ケース時間は、要求される実効アドレスの計算(CCea)および残りの操作に対する命令キャッシュ・ケース時間(CCop)に区分されます。すべての命令およびアドレッシング・モードに対する命令キャッシュ・ケース時間を「11.6　命令実行時間表」に記載しています。

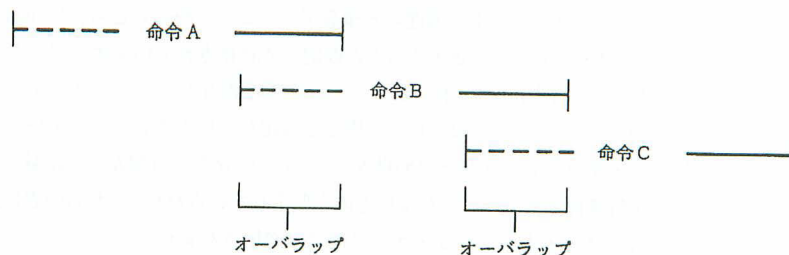



図11-2　命令の同時実行

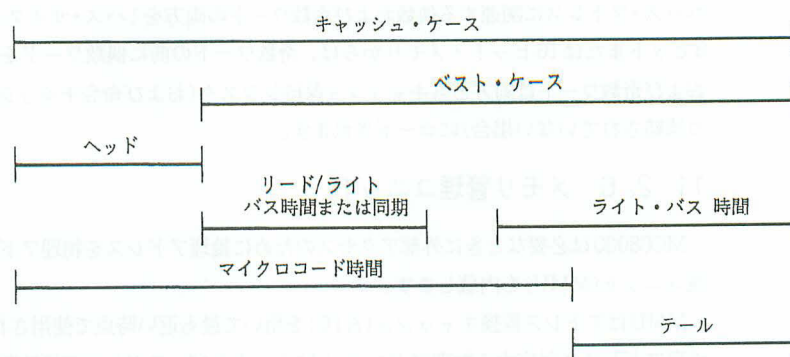



図11-3 命令オーバラップ時間の詳細

11. 3. 2 オーバラップおよびベスト・ケース

オーバラップとは、ある命令が前の命令と並行して実行される時間をクロック周期で測定したものです。図11-2で、命令AおよびBの一部は同時に実行されます。オーバラップ時間は2つの命令に対する全体的な実行時間を低減します。同様に、命令Bと命令Cの間のオーバラップ期間は、これら2つの命令の全体的な実行時間を低減します。

各命令は合計オーバラップ時間に寄与します。図11-2に示すように、命令Bの実行開始時における時間の一部は、命令Aの実行時間の終わりの部分とオーバラップすることができます。この期間を命令Bのヘッドとよびます。命令Bの開始部分とオーバラップ可能な命令Aの終わりの時間部分を命令Aのテールとよびます。命令Aおよび命令Bの間の合計オーバラップ時間は、命令Aのテールまたは命令Bのヘッドのいずれか小さいほうからなります。ヘッドおよびテール時間については、「11. 6 命令実行時間表」にある命令実行時間表を参照してください。

図11-3に実効アドレス計算(CCea)または操作(CCop)のいずれかに対する命令-キャッシュ・ケース時間を構成する諸要素の実行時間の関係を示します。図11-2において、命令Bのベスト・ケースの実行時間は、命令Bおよび命令Aに対する命令-キャッシュ・ケース時間がオーバラップし、命令Bのヘッドが命令Aのテールと完全にオーバラップするときに発生します。

命令オーバラップの特質、および一部の命令のヘッドがそれらの命令の合計命令－キャッシュ・ケース時間に等しいという事実により、正味ゼロの実行時間が可能です。ある命令の実行時間は、前の命令とのオーバラップによって完全に吸収されてしまう場合があります。

11. 3. 3 平均ノー・キャッシュ・ケース

ある命令に対する平均ノー・キャッシュ・ケース(NCC)時間は、マイクロコードが実行するのに要する時間、およびすべての外部バス・アクティビティに要する時間を考慮しています。この時間はキャッシュ・ミスおよび関連する命令プリフェッチが両方とも、2つの命令のプリフェッチにつき、1外部バス・サイクルを要するものと仮定して計算されています。「11. 2. 2 命令パイプ」を参照してください。平均ノー・キャッシュ・ケース時間もオーバラップがないものと仮定しています。すべてのバス・サイクルは2クロック周期と仮定しています。命令および実効アドレス計算に対する平均ノー・キャッシュ・ケース時間を「11. 6 命令実行時間表」に記載します。ノー・キャッシュ・ケース時間はオーバラップがないものと仮定しているため、これらの表に記載するヘッドおよびテール値は、ノー・キャッシュ・ケース値には適用されません。

実際のノー・キャッシュ・ケース時間は、命令に関連するプリフェッチのアラインメントによって異なるため、両方のアラインメントのケースを考慮して、表に示す値は奇数ワードに整列したケー

スおよび偶数ワードに整列したケースを平均したものです(端数を整数クロック数に切上げ)。同様に、プリフェッチ・バス・サイクル数はこれら2つのケースの平均を整数バス・サイクル数に切り上げたものです。

実行時間に関する命令のアラインメントの影響を次の例で説明します。「11. 6　命令実行時間表」に引用した仮定を適用しています。すべてのアクセスでデータ・キャッシュおよび命令キャッシュ・ミスが発生しています。

命　令

1. MOVE.L　　(d_{16}An, Dn), Dn
2. CMPI.W　#<data>.W,(d_{16}, AN)

命令ストリームは、32ビット・メモリ内で次のように偶数アラインメントによって位置決めされます。

アドレス		
n	MOVE	EA Ext
n+4	d_{16}	CMPI
n+8	#⟨data.W⟩	d_{16}
n+12	...	...

図11-4はある命令ストリームの偶数アラインメントに対するプロセッサのアクティビティを示します。ここでは、外部バス、バス・コントローラ、およびシーケンサのアクティビティを示します。

図11-5は奇数アラインメントに対するプロセッサ・アクティビティを示します。命令ストリームは、32ビット・メモリ内で次のように奇数アラインメントによって位置決めされます。

アドレス		
n	...	MOVE
n+4	EA Ext	d_{16}
n+8	CMPI	#⟨data.W⟩
n+12	d_{16}	...

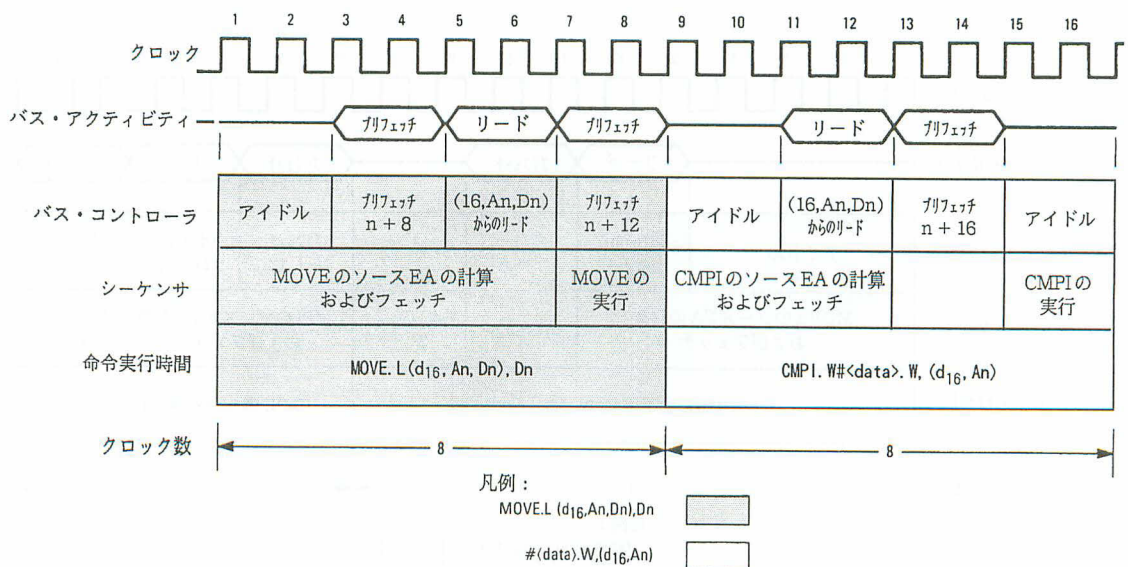



図11-4　プロセッサのアクティビティ——偶数アラインメント

2つのアラインメントを比較すると、MOVE命令の実行時間は偶数アラインメントに対し8クロック、奇数アラインメントに対し10クロックであり、平均9クロックです。「11. 6. 6　MOVE命令」にある表および「11. 6. 1　実効アドレスのフェッチ」にある表を見ると、平均ノー・キャッシュ・ケース時間は2＋7＝9クロックです。平均ノー・キャッシュ・ケース時間が7クロックのCMPI命令についても、同様な計算を行なうことができます。

多くの場合、最大ノー・キャッシュ・ケースの実行時間よりも、平均ノー・キャッシュ・ケースの実行時間のほうが、命令ストリームの実際の実行時間に近い概算値が得られます。上記の例の2つの命令の合計実行時間は、偶数および奇数アラインメントの両方とも16クロックです。与えられた命令の平均ノー・キャッシュ・ケースの実行時間を加算すると、これも16クロック(9＋7＝16)になります。ここでもう一度、NCC時間はオーバラップがないものと仮定していることを思い出してください。したがって、命令ストリームの実際の時間は、NCC時間を加算して得られる値よりも小さくなります。ノー・キャッシュ・ケースに対するウエイト・ステートの影響における要因については、「11. 5　ウエイト・ステートの影響」を参照してください。

11. 3. 4　実際の命令-キャッシュ・ケースの実行時間計算

ある命令の全体の実行時間は、その前後の命令とのオーバラップによって異なります。したがって、命令の概算実行時間を計算するには、評価する全体のコード・シーケンスを一括して分析しなければなりません。ある命令シーケンスに対する実際の命令-キャッシュ・ケースの実行時間(「11.6 命令実行時間表」に記載する仮定を用いて)を求めるために、表に記載されている命令-キャッシュ・ケース時間を使用し、そして全体のシーケンスに対して、適切なオーバラップを減算しなければなりません。この計算の式は次のとおりです。

$CC_1 + [CC_2 - \min(H_2, T_1)] + [(CC_3 - \min(H_3, T_2)] + \cdots\cdots$ 式 (11-1)

ここで、

CC_n は命令の命令-キャッシュ・ケース時間

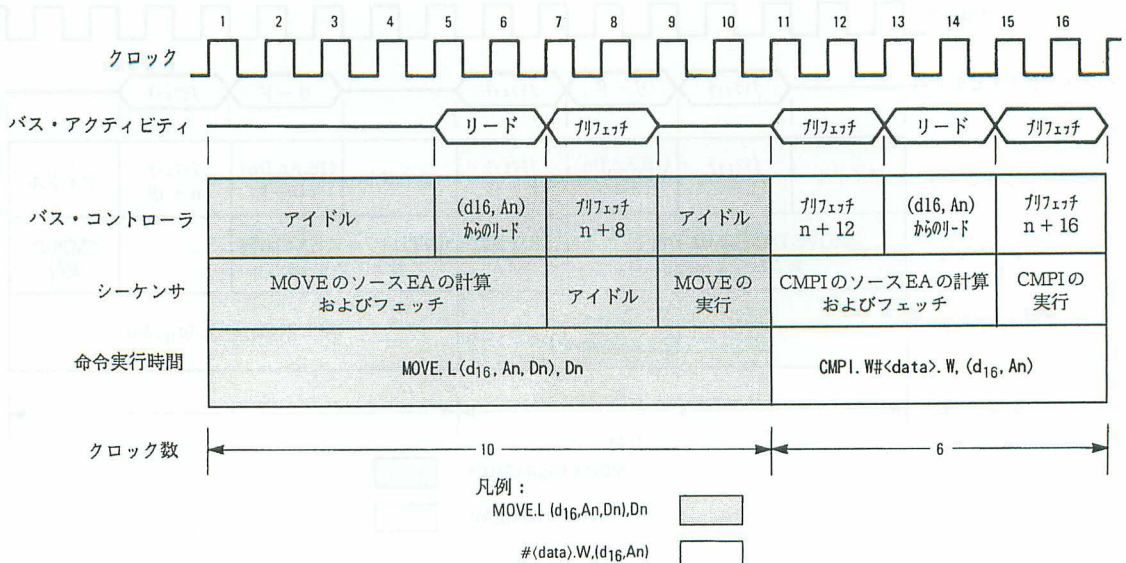



図11-5　プロセッサのアクティビティ――偶数アラインメント

T_n は命令のテール時間

H_n は命令のヘッド時間

min(a, b)はパラメータaおよびbの最小値

ほとんどの命令に対する命令-キャッシュ・ケース時間は、操作の命令-キャッシュ・ケース時間(CCop)にオーバラップした実効アドレス計算(CCea)の命令-キャッシュ・ケース時間で構成されています。具体的な式は次のとおりです。

$CCea_1 + [CCop_1 - \min(Hop_1, Tea_1)] + [CCea_2 - \min(Hea_2, Top_1)] +$
$[CCop2 - \min(Hop_2, Tea_2)] + [CCea_3 - \min(Hea_3, Top_2)] + \cdots\cdots\cdots\cdots$ 式 (11-2)

ここで、

$CCea_n$ は命令-キャッシュ・ケースの実効アドレス時間

$CCop_n$ は命令の操作部分に対する命令-キャッシュ・ケース時間

Tea_n は命令の実効アドレスのテール時間

Hop_n は命令の操作部分のヘッド時間

Top_n は命令の操作部分のテール時間

Hea_n は命令の実効アドレスのヘッド時間

min(a, b)はパラメータaおよびbの最小値

実効アドレスの命令-キャッシュ・ケース、ヘッド、およびテール(CCea、Hea、およびTea)をCCop、Hop、およびTopとオーバラップさせる必要のある命令は、「11. 6　命令実行時間表」のところで脚注を付けてあります。

命令ストリームに対する実際の命令-キャッシュ・ケースの実行時間は、式11-1または式11-2を用いて計算することができます。式11-1は、実効アドレスの命令-キャッシュ・ケース、ヘッドおよびテールが必要なときに使用します。

命令-キャッシュ・ケースの実行時間を計算するのに、式11-1を必要とする一連の命令を使用する例は次のとおりです。「11. 6　命令実行時間表」に記載する仮定を適用しています。

命　令

1.　ADD.L　A1, D1

2.　SUBA.L　D1, A2

「11. 6. 8　算術/論理演算命令」の実行時間表を見てみると、ADD.L A1、D1およびSUBA.L D1、A2に対するヘッド、テール、および命令－キャッシュ・ケース(CC)時間が分かります。いずれかの命令に対して実効アドレス時間を加算するよう指示する脚注はありません。両方の命令とも、レジスタ・オペランドしか使用しないため、実効アドレスの計算時間を加算する必要はありません。したがって、両方に対して式11-1を使用することができます。

	ヘッド	テール	CC
1. ADD.L　A1, D1	2	<u>0</u>	2
2. SUBA.L　D1, A2	<u>4</u>	0	4

注：下線が付いた数字は、次の公式でヘッドとテールを比較するための代表的なパターンを示します。

次の計算では式11-1を使用しています。

実行時間 $= CC_1 + [CC_2 - \min(H_2, T_1)]$
$= 2 + [4 - \min(4, 0)]$
$= 2 + [4 - 0]$
＝6クロック

該当する表から、実効アドレス計算時間を加算する必要のある命令は、式11-2を使用して実際の

CC時間を計算しています。CCea、Hea、およびTeaの値は、指示されるとおり、該当する実効アドレス表(実効アドレスのフェッチ、イミディエイト実効アドレスのフェッチ、実効アドレスの計算、イミディエイト実効アドレスの計算、または実効アドレスのジャンプのいずれか)から抽出し、公式11-2に代入しなければなりません。以下の命令は、最後のものを除いてすべて式11-2を必要とします。最後の命令は、式11-1を使用します。

命令
1. ADD.L　　−(A1), D1
2. AND.L　　D1,([A2])
3. MOVE.L　　(A6), (8, A1)
4. TAS　　(A3)＋
5. NEG　　D3

「11.6　命令実行時間表」の該当する操作および実効アドレス表を使用すると、以下のような結果になります。

	ヘッド	テール	CC
1. ADD.L −(A1), D1			
実効アドレスのフェッチ(fea)−(An)	2	2	4
ADD EA, DM	0	0	2
2. AND.L D1, ([A2])			
fea ([B])	4	0	10
AND Dn, EA	0	1	3
3. MOVE.L(A6), (8, A1)			
fea(An)	1	1	3
MOVE ソース, (d_{16}, An)	2	0	4
4. TAS(A3)＋			
実効アドレスの計算 Cea (An)＋	0	0	2
TAS Mem	3	0	12
5. NEG D3	2	0	2

次の計算では式11-2および11-1を使用します。

$$\begin{aligned}
\text{実行時間} &= CCea_1 + [CCop_1 - \min(Hop_1, Tea_1)] + [CCea_2 - \min(Hea_2, Top_1)] + \\
&\quad [CCop_2 - \min(Hop_2, Tea_2)] + [CCea_3 - \min(Hea_3, Top_2)] + \\
&\quad [CCop_3 - \min(Hop_3, Tea_3)] + [CCea_4 - \min(Hop_4, Top_3)] + \\
&\quad [CCop_4 - \min(Hop_4, Top_3)] + [CCop_5 - \min(Hop_5, Top_4)] \\
&= 4 + [2 - \min(0,2)] + [10 - \min(4,0)] + [3 - \min(0,0)] + [3 - \min(1,1)] + \\
&\quad [4 - \min(2,1)] + [2 - \min(0,0)] + [12 - \min(3,0)] + [2 - \min(2,0)] \\
&= 4 + 2 + 10 + 3 + 2 + 3 + 2 + 12 + 2 \\
&= 40 \text{クロック周期}
\end{aligned}$$

なお、最後の命令は実効アドレス(ea)の加算がないため、式11-2が必要ありませんでした。したがって、式11-1を使用しています。

CCop5 − min (Hop5, Top4)

イミディエイト実効アドレスのフェッチ(fiea)またはイミディエイト実効アドレスの計算(ciea)表を使用するときには、実行時間計算においてデータのサイズが重要です。各実効アドレスに対し、

ワード・データは#<data>.W、ロング・ワード・データは#<data>.Lで表わしてあります。

いくつかの実効アドレス・タイプのヘッドの合計は、実効アドレス計算の間拡張され、操作のヘッドを含んでいます。これらの実効アドレス計算は、次のようにヘッドの欄に記入されています。

X＋op　head

ここで、

Xは実効アドレスだけのヘッドです。

fiea表およびX＋op　headの表記の使い方は次のとおりです。

命令

1. EORI.W　　#$400, －(A1)
2. ADDI.L　　#$6000FF, D1

	ヘッド	テール	CC
1. EORI.W #$400,－(A1)			
fiea #<data>.W,－(An)	2	2	4
EORI #<data>,Mem	0	1	3
2. ADDI.L #$6000FF,D1			
fiea #<data>.L,D1	4＋op head	0	4
	6	0	4
ADDI #<data>,Dn	2(op head)	0	2

次の計算は式11-2を使用しています。

$$
\begin{aligned}
\text{実行時間} &= CCea_1+[CCop_1-\min(Hop_1,Tea_1]+[CCea_2-\min(Hea_2,Top_1)]+ \\
&\quad [CCop_2-\min(Hop_2,Tea_2)] \\
&= 4+[3-\min(0,2)]+[4-\min(6,1)]+[2-\min(2,0)] \\
&= 4+3+3+2 \\
&= 12\ \text{クロック周期}
\end{aligned}
$$

fiea #<data>.L、D1、4＋op headのヘッドに対して、その結果生じたヘッド6はそのフェッチの命令-キャッシュ・ケース時間よりも大きいことに注意してください。その部分の実行時間は負数(たとえば、4－min(6,6)＝－2)になります。この結果は、フェッチが完全にオーバラップし、同じテールで操作が部分的にオーバラップされるため、正しい実行時間です。操作実行時間の計算を変更する必要はありません。

多くの2ワード命令(たとえば、MULU.L、DIV.L、BFSETなど)は、実行時間の計算にイミディエイト実効アドレスのフェッチ(fiea)時間、またはイミディエイト実効アドレスの計算(ciea)時間を含んでいます。これらの計算には、ワード長のイミディエイト・データ(#<data>.W)に対する実行時間を使用します。命令にソースおよびデスティネーションがある場合は、ソースEAを使用してテーブル・ルックアップを行ないます。命令が単一オペランドの場合は、そのオペランドの実効アドレスを使用します。

次の例には、「11. 6　命令実行時間表」のイミディエイト実効アドレスのフェッチおよびイミディエイト実効アドレスの計算表を引用する複数ワード命令が含まれています。

命令

1. MULU.L　　(D7),D1 : D2
2. BFCLR　　$6000 {0 : 8}
3. DIVS.L　　#$10000, D3 : D4

	ヘッド	テール	CC
1. MULU.L (D7),D1:D2			
fiea #<data>.W,Dn	2+op head	0	2
	4	0	2
MUL.L EA, Dn	2(op head)	0	44
2. BFCLR $6000{0:8}			
fiea #<data>.W,$XXX.W	4	2	6
BFCLR Mem(<5 bytes)	6	0	14
3. DIVS.L #$10000,D3:D4			
fiea #<data>.W,#<data>.L	6+op head	0	6
	6	0	6
DIVS.L EA,Dn	0(op head)	0	90

式11-2を用いて次の式を計算します。

$$\begin{aligned}\text{実行時間} &= CCea_1+[CCop_1-\min(Hop_1,Tea_1)]+[CCea_2-\min(Hea_2,Top_1)]+\\&\quad [CCop_2-\min(Hop_2,Tea_2)]+[CCea_3-\min(Hea_3,Top_2)]+\\&\quad [CCop_3-\min(Hop_3,Tea_3)]\\&=2+[44-\min(2,0)]+[6-\min(4,0)]+[14-\min(6,2)]+[6-\min(6,0)]+\\&\quad [90-\min(0,0)]\\&=2+44+6+12+6+90\\&=160\ \text{クロック周期}\end{aligned}$$

注：このCC時間は、MULU.LおよびDIVS.Lに最大時間を与えているため最大になります。

11．4　データ・キャッシュの効果

命令で要求されるデータ・アクセスがデータ・キャッシュの中にあるときには、これらのオペランドを読み出すのにバス・サイクルは必要なく、その命令の実行時間を最小にすることができます。しかし、ライト・アクセスは、データ・キャッシュがライト・スルー・キャッシュであるため、常にバス・サイクルを必要とします。

オペランド・リード・アクセスに関するデータ・キャッシュの効果は、次のように実際の命令の実行時間に分解することができます。

「11 .6　命令実行時間表」にある実効アドレスのフェッチ表、またはイミディエイト実効アドレスのフェッチ表のいずれかに対応するデータ・フェッチで、データ・キャッシュのヒットが起こったときは、次の規則が適用されます。

1a. $Tail_t=0$の場合：　実行時間に変化なし。

1b. $Tail_t=1$の場合：　$Tail=Tail_t-1$
　　$CC=CC_t-1$

1c. $Tail_t>1$の場合：　$Tail=Tail_t-(Tail_t-1)=1$
　　$CC=CC_t-(Tail_t-1)$

ここで、$Tail_t$およびCC_tはテーブル中に記載された値です。

2. EAモードがメモリ間接(2回のデータ・リード)の場合、テールおよびCC時間は1回のデータ・リードとして計算します。

注：CCopにオペランドのフェッチがある命令や操作(たとえば、BFFFOおよびCHK2)の実行時間に、データ・キャッシュのヒットを含めて計算することは容易ではありません。このような

CCopでのデータ・キャッシュのヒットの効果は、計算では無視されています。

RMCサイクル(たとえば、TASおよびCAS)はデータ・キャッシュ・リードを強制的にミスさせます。したがって、データ・キャッシュのヒットはこれらの命令には、影響を与えません。

次の例はデータ・キャッシュのヒットを想定しています。データ・キャッシュのヒットに訂正された行は太字で印字されています。これらの行を使用して命令-キャッシュ・ケースの実行時間を計算します。前の規則を参照してください。

命令

```
1.  ADD.L     -(A1),D1
2.  AND.L     D1,([A2])
3.  MOVE.L    (A6),(8,A1)
4.  TAS       (A3)
```

	ヘッド	テール	CC
1. ADD.L −(A1),D1			
Fetch Effective Address			
fea −(An)	2	2-1	4-1(1/0/0)
***1c**	**2**	**1**	**3(1/0/0)**
***ADD EA,Dn**	**0**	**0**	**2(0/0/1)**
2. AND.L D1,([A2])			
***1a & 2 fea ([B])**	**4**	**0**	**10(2/0/0)**
***AND Dn,EA**	**0**	**1**	**3(0/0/1)**
3. MOVE.L (A6),(8,A1)			
fea (An)	1	1-1	3-1(1/0/0)
***1b**	**1**	**0**	**2(1/0/0)**
***MOVE Source, (d_{16},An)**	**2**	**0**	**4(0/0/1)**
4. TAS (A3)+			
***Cea (An)+**	**0**	**0**	**2(0/0/0)**
***TAS Mem**	**0**	**0**	**12(1/0/1)**

＊データ・キャッシュのヒットを訂正

注：実行時間に対するデータ・キャッシュのヒットの影響を計算するために、CCカラムの命令アクセス数にオペランド読出しおよび書込み回数を含めておけばわかりやすくなります。

$$
\begin{aligned}
\text{実行時間} = {} & CCea_1 + [CCop_1 - \min(Hop_1, Tea_1)] + [CCea_2 - \min(Hea_2, Top_1)] + \\
& [CCop_2 - \min(Hop_2, Tea_2)] + [CCea_3 - \min(Hea_3, Top_2)] + \\
& [CCop_3 - \min(Hop_3, Tea_3)] + [CCea_4 - \min(Hea_4, Top_3)] + \\
& [CCop_4 - \min(Hop_4, Tea_4)] \\
= {} & 3 + [2 - \min(0,1)] + [10 - \min(4,0)] + [3 - \min(0,0)] + [2 - \min(1,1)] + \\
& [4 - \min(2,0)] + [2 - \min(0,0)] + [12 - \min(0,0)] \\
= {} & 3 + 2 + 10 + 3 + 1 + 4 + 2 + 12 \\
= {} & 37\ \text{クロック周期}
\end{aligned}
$$

11.5 ウエイト・ステートの影響

システム設計の制約によって、メモリ・サイクルにウエイト・ステートを挿入しなければならないことがあります。バスまたはメモリ・デバイスが多くのウエイト・ステートを必要とするときは、命令実行時間が増加します。しかし、1つまたは2つのウエイト・ステートだけなら、命令の実行時間にほとんど影響を与えません。1または2ウエイト・ステートで見られる影響は、バスのアイドル時間が減ることだけです。

データ・アクセス・ステートに対するウエイト・ステートの影響は、命令-キャッシュ・ケースの実行時間で説明します。

データ・アクセスにウエイト・ステートの影響を反映させるには、次のようにします。

1a. オペランドのリードを含む非メモリ間接実効アドレス実行時間に対しては、ウエイト・ステート数(クロック単位)をテールおよび命令－キャッシュ・ケース(CC)時間に加算します。ヘッドには影響はありません。

1b. < ea >の計算表を使用し、1回のデータ・リード(アドレス・フェッチのためのもの)だけをもつメモリ間接実効アドレス実行時間に対しては、CC時間にだけウエイト・ステート数を加算します。ヘッドおよびテールには影響はありません。

1c. 2つのデータ・リード(アドレス・フェッチのためのもの)をもつメモリ間接実効アドレスの実行時間(< ea >のフェッチ)に対しては、CC時間に2回のリードに対するウエイト・ステート数を加算します。1回のデータ・リードに対するウエイト・ステート数をテールに加算します。ヘッドには影響はありません。

2a. データ・リードを含む操作実行時間(たとえば、BFFFOおよびTAS)に対しては、CC時間にウエイト・ステート数を加算します。ヘッドおよびテールのどちらにも影響はありません。

注：MOVEM命令のCC実行時間およびテールは、データ・リードおよびライトの両方に対する特別なケースです。ウエイト・ステートの関数で表わす、CC実行時間およびテール両方に対する公式は、「11. 6. 7　特殊目的のMOVE命令」にある表の中に脚注が付けてあります。

2b. 操作に2回以上のデータ・リードが含まれる場合は、CC時間にすべてのリードに対する合計ウエイト・ステート数を加算します。ヘッドおよびテールのどちらにも影響はありません。上記の注を参照してください。

3a. データ・ライトを含む操作実行時間に対しては、テールおよびCC時間にウエイト・ステート数を加算します。ヘッドには影響はありません。上記の注を参照してください。

3b. 操作に2回以上のライトが含まれる場合は、テールは1回のライトに対するウエイト・ステート数分だけ増加します。CC実行時間はすべてのライトに対する合計ウエイト・ステート数分だけ増加します。上記の注を参照してください。

次の例は、2ウエイト・ステート(4クロック・リードおよびライト)をもつ指定された命令ストリームのキャッシュ・ケースの命令実行時間を計算します。ウエイト・ステートについて訂正した行は太字で印字されており、命令実行時間を計算するのに使用します。上記の規則を参照してください。

命令

1. MOVE.L　($800,A2,D3),(A5,D2)
2. ADD.L　D1,([$30,A4])
3. BFCLR　($20,A5){1:5} — (<5 bytes)
4. BFTST　($10,A3,D3){31:31} — (5 bytes)
5. MOVEM　([A1,D1]),A1-A4 — 4 registers

ウエイト・ステート＝2

	ヘッド	テール	CC
1. MOVE.L ($800,A2,D3),(A5,D2)			
fea (d_{16},An,Xn)	4	0+2	6+2(1/0/0)
***1a**	**4**	**2**	**8(1/0/0)**
MOVE Source,(B)	4	0+2	8+2(0/0/1)
***3a**	**4**	**2**	**10(0/0/1)**
2. ADD.L D1,([$30,A4])			
fea ([d_{16},B])	4	0+2	12+4(2/0/0)
***1c**	**4**	**2**	**16(2/0/0)**
ADD Dn,EA	0	1+2	3+2(0/0/1)
***3a**	**0**	**3**	**5(0/0/1)**
3. BFCLR ($20,A5){1:5}			
***ciea #<data>.W,(d_{16},An) Single EA Format**	**10**	**0**	**4(0/0/0)**
BFCLR Mem (< 5 bytes)	6	0+2	14+4(1/0/1)
***2a & 3a**	**6**	**2**	**18(1/0/1)**
4. BFTST ($10,A3,D3){31:31}			
***ciea (d_{16},An,Xn)**	**14**	**0**	**8(0/0/0)**
BFTST Mem (5 bytes)	6	0	14+4(2/0/0)
***2b**	**6**	**0**	**18(2/0/0)**
5. MOVEM ([A1,D1]),A1-A4			
ciea ([B])	6	0	12+2(1/0/0)
***1b**	**6**	0	**14(1/0/0)**
MOVEM EA,RL	2	0	24+0(4/0/0)
***2a & 2b**	**2**	**0**	**24(4/0/0)**

＊ウエイト・ステートを訂正

注：実行時間に対するウエイト・ステートの影響を計算するために、CCカラムの命令アクセス数にオペランド読出しおよび書込み回数を含めておけば分かりやすくなります。

式11-2を用いて次のとおり計算します。

$$
\begin{aligned}
\text{実行時間} &= CCea_1+[CCop_1-\min(Hop_1,Tea_1)]+[CCea_2-\min(Hea_2,Top_1)]+\\
&\quad [CCop_2-\min(Hop_2,Tea_2)]+[CCea_3-\min(Hea_3,Top_2)]+\\
&\quad [CCop_3-\min(Hop_3,Tea_3)]+[CCea_4-\min(Hea_4,Top_3)]+\\
&\quad [CCop_4-\min(Hop_4,Tea_4)]+[CCea_5-\min(Hea_5,Top_4)]+\\
&\quad [CCop_5-\min(Hop_5,Tea_5)]\\
&= 8+[10-\min(4,2)]+[16-\min(4,2)]+\\
&\quad [5-\min(0,2)]+[4-\min(10,3)]+[18-\min(6,0)]+[8-\min(14,2)]+\\
&\quad [18-\min(6,0)]+[14-\min(6,0)]+\\
&\quad [24-\min(2,0)]\\
&= 8+8+14+5+1+18+6+18+14+24\\
&= 116\ \text{クロック周期}
\end{aligned}
$$

次の例は、「11. 4　データ・キャッシュの効果」から引用した、1サイクル当たり2ウエイト・ステート(4クロック・リード/ライト)のデータ・キャッシュのヒット例です。データ・キャッシュおよび命令キャッシュでヒットが起こったものと仮定しています。各実行時間に対して3行を示します。最初の行は該当する表からの実行時間です。2行目はデータ・キャッシュのヒットに対して調整した実行時間です。そして3行目は、リード操作がキャッシュでヒットし、遅延が生じてないため、ライト操作だけにウエイト・ステートを加算したものです。各実行時間に対する3行目を使用して命令キャッシュの実行時間を計算します。これは太字で示してあります。

命令

1. ADD.L　　−(A1),D1
2. AND.L　　D1,([A2])
3. MOVE.L　　(A6),(8,A1)
4. TAS　　(A3)+

	ヘッド	テール	CC
1. ADD.L −(A1),D1			
fea −(An)	2	2	4(1/0/0)
*	2	1	3(1/0/0)
**	**2**	**1**	**3(1/0/0)**
ADD.L EA,Dn	0	0	2(0/1/0)
*	0	0	2(0/1/0)
**	**0**	**0**	**2(0/1/0)**
2. AND.L D1,([A1])			
fea ([B])	4	0	10(1/0/0)
*	4	0	10(1/0/0)
***	**4**	**0**	**12(1/0/0)**
AND Dn,EA	0	1	3(0/0/1)
*	0	1	3(0/0/1)
**	**0**	**3**	**5(0/0/1)**
3. MOVE.L (A6),(8,A1)			
fea (An)	1	1	3(1/0/0)
*	1	0	2(1/0/0)
**	**1**	**0**	**2(1/0/0)**
MOVE Source,(d_{16},An)	2	0	4(0/0/1)
*	2	0	4(0/0/1)
**	**2**	**2**	**6(0/0/1)**
4. TAS (A3)+			
Cea (An)	0	0	2(0/0/0)
*	0	0	2(0/0/0)
**	**0**	**0**	**2(0/0/0)**
TAS Mem	3	0	12(1/0/1)
*	3	0	12(1/0/1)
**	**3**	**0**	**14(1/0/1)**

注：＊データ・キャッシュ・ヒットに対して訂正

＊＊ウエイト・ステートに対しても訂正(データ・ライトのみ)。

＊＊＊アドレス・フェッチでデータ・キャッシュのヒットが起こらなかったと仮定

式11-2を用いて次のとおり計算します。

$$
\begin{aligned}
\text{実行時間} &= CCea_1 + [CCop_1 - \min(Hea_1, Top_1)] + [CCea_2 - \min(Hea_2, Top_1)] + \\
&\quad [CCop_2 - \min(Hop_2, Tea_2)] + [CCea_3 - \min(Hea_3, Top_2)] + \\
&\quad [CCop_3 - \min(Hop_3, Tea_3)] + [CCea_4 - \min(Hea_4, Top_3)] + \\
&\quad [CCop_4 - \min(Hop_4, Tea_4)] \\
&= 3 + [2 - \min(0,1)]m + [12 - \min(4,0)] + \\
&\quad [5 - \min(0,0)] + [2 - \min(1,3)] + \\
&\quad [6 - \min(2,0)] + [2 - \min(0,2)] + \\
&\quad [14 - \min(3,0) \\
&= 3 + 2 + 12 + 5 + 1 + 6 + 2 + 14 \\
&= 45\ \text{クロック周期}
\end{aligned}
$$

平均ノー・キャッシュ・ケースに対しても、同様の分析を行なうことができます。平均ノー・キャッシュ・ケース時間は、1バス・サイクル当たり2クロック周期(つまり、ノー・ウエイト・ステート)を仮定するため、表に記載された実行時間はウエイト・ステートをもつシステムには、直接適用されません。命令またはWウエイト・ステート付き実効アドレスに対する平均ノー・キャッシュ・ケース時間を推定するには、次の公式を使用します。

$$NCC = NCC_t + (\text{データ・リードおよびライト数}) \cdot W + (\text{最大命令アクセス数}) \cdot W$$

ここで、

NCC_t は該当する表からのノー・キャッシュ・ケースの実行時間値

データ・リード数、データ・ライト数、および最大命令アクセス数は、該当する表に記載されています。

この公式から得られた平均ノー・キャッシュ・ケースの実行時間は、最大命令アクセス数(表中の値は常に切上げ)を使用しオーバラップがないものと仮定しているため、実際のノー・キャッシュ・ケースの実行時間に等しいか、あるいはそれ以上になります。

11.6　命令実行時間表

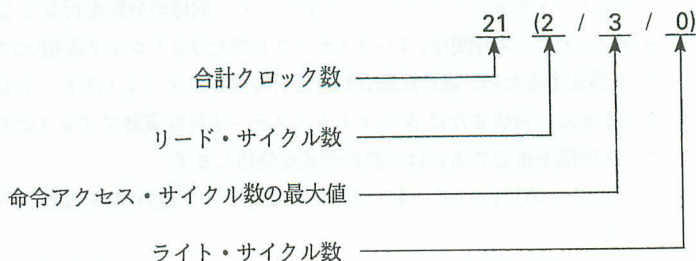
以下の各表に示す命令実行時間には、次の仮定事項が含まれています。

- すべてのメモリ・アクセスは2クロック・バス・サイクルおよびノー・ウエイト・ステートで発生する。
- システム・スタックを含め、メモリ内のすべてのオペランドは、ロング・ワードに整列している。
- MC68030とシステム・メモリ間の交信には、32ビットのバスを使用する。
- データ・キャッシュはイネーブルされていない。
- 例外は発生しない(指定されている場合を除く)。
- すべてのバス・サイクルに対して必要なアドレス変換は、アドレス変換キャッシュに存在している。

各命令および実効アドレスに対しては、次の4つの値が示してあります。

1. ヘッド
2. テール
3. 命令−キャッシュ・ケース(CC): 命令がキャッシュに存在するがオーバラップがないとき
4. 平均ノー・キャッシュ・ケース(NCC): 命令がキャッシュに存在しないか、キャッシュがディセーブルされていて、命令のオーバラップがないとき

オペランドのサイズが影響を与える唯一の場合は、イミディエイト・オペランド付き命令、そしてADDAおよびSUBA命令です。特に規定されていないかぎり、イミディエイト・バイトおよびイミディエイト・ワード・オペランドの実行時間は同じです。

命令実行時間表の命令-キャッシュ・ケースおよび平均ノー・キャッシュ・ケースのカラムには、4組の数値が記載されており、そのうち3つはかっこで囲まれています。外側に記載されている数値は、あるキャッシュ・ケースおよび命令の合計クロック数です。かっこ内の最初の数値は、その命令で実行されるオペランド・リード・サイクル数です。かっこ内の2番目の数値は、命令パイプを充てんしておくためのプリフェッチを含むバス・サイクル数の最大数を示します。2番目の数値は、奇数ワード整列ケースと偶数ワード整列の平均(整数バス・サイクル数に切り上げられる)ですので、常に実際のバス・サイクル数(2命令当たり1バス・サイクル)より大きいかそれと等しくなります。かっこ内の3番目の数値は、その命令によって実行されるライト・サイクル数です。命令実行時間表からの一例を次に示します。

この例の命令のバス・アクティビティ・クロックと内部クロック(バス・アクティビティでオーバラップされない)の合計クロック数は、次式によって計算されます。

(2リード・2クロック/リード)+(3命令アクセス・2クロック/アクセス)+
(0ライト・2クロック/ライト)=バス動作の10クロック

合計クロック21－10バス動作クロック＝11内部クロック

ここでとりあげた例は、ノー・キャッシュ・ケースの‘実効アドレスのフェッチ’時間から引用したものです。このアドレッシング・モードは([d32, B], I, d32)です。命令-キャッシュ・ケースでの同じアドレッシング・モードの実行時間は18(2/0/0)です。命令-キャッシュ・ケースの実行時間に対しては、キャッシュがイネーブルされていて、シーケンサが命令ワードを読み出すために外部メモリにアクセスする必要がないため、命令アクセスは必要ありません。

最初の5つの実行時間表は、実効アドレスおよびイミディエイト・オペランドの計算とフェッチだけを扱っています。残りの表には、命令およびオペランドの実行時間が記載されています。命令によっては、対応する命令実行時間にない特殊なアドレッシング・モードを使用するものがあります。このような場合、実行時間の計算にほかの表が必要であることを示す脚注が付けられています。すべてのリードおよびライト・アクセスは、2クロック周期と仮定されています。

11.6.1　実効アドレスのフェッチ(fea)

実効アドレスのフェッチの表には、プロセッサが指定された実効アドレスを計算してフェッチするのに必要なクロック周期数が示してあります。実効アドレスはフォーマットで分類されています(「2.5　実効アドレスのエンコーディングの概要」参照)。命令-キャッシュ・ケースおよびノー・キャッシュ・ケースでは、合計クロック・サイクル数は、かっこの外側に記入されています。リード、プリフェッチ、およびライト・サイクル数は、かっこ内に(r/p/w)の形式で記載されています。これらは、合計クロック・サイクル数に含まれています。

すべての実行時間のデータは、2クロックのリードおよびライトを想定しています。

アドレス・モード	ヘッド	テール	Iキャッシュ・ケース	ノー・キャッシュ・ケース
単一実効アドレス命令のフォーマット				
%　Dn	—	—	0 (0/0/0)	0 (0/0/0)
%　An	—	—	0 (0/0/0)	0 (0/0/0)
(An)	1	1	3 (1/0/0)	3 (1/0/0)
(An)+	0	1	3 (1/0/0)	3 (1/0/0)
−(An)	2	2	4 (1/0/0)	4 (1/0/0)
(d_{16},An) or (d_{16},PC)	2	2	4 (1/0/0)	4 (1/1/0)
(xxx).W	2	2	4 (1/0/0)	4 (1/1/0)
(xxx).L	1	0	4 (1/0/0)	5 (1/1/0)
#⟨data⟩.B	2	0	2 (0/0/0)	2 (0/1/0)
#⟨data⟩.W	2	0	2 (0/0/0)	2 (0/1/0)
#⟨data⟩.L	4	0	4 (0/0/0)	4 (0/1/0)
簡潔フォーマット拡張ワード				
(d_8,An,Xn) or (d_8,PC,Xn)	4	2	6 (1/0/0)	6 (1/1/0)
全フォーマット拡張ワード				
(d_{16},An) or (d_{16},PC)	2	0	6 (1/0/0)	7 (1/1/0)
(d_{16},An,Xn) or (d_{16},PC,Xn)	4	0	6 (1/0/0)	7 (1/1/0)
([d_{16},An]) or ([d_{16},PC])	2	0	10 (2/0/0)	10 (2/1/0)
([d_{16},An],Xn) or ([d_{16},PC],Xn)	2	0	10 (2/0/0)	10 (2/1/0)
([d_{16},An],d_{16}) or ([d_{16},PC],d_{16})	2	0	12 (2/0/0)	13 (2/2/0)
([d_{16},An],Xn,d_{16}) or ([d_{16},PC],Xn,d_{16})	2	0	12 (2/0/0)	13 (2/2/0)
([d_{16},An],d_{32}) or ([d_{16},PC],d_{32})	2	0	12 (2/0/0)	14 (2/2/0)
([d_{16},An],Xn,d_{32}) or ([d_{16},PC],Xn,d_{32})	2	0	12 (2/0/0)	14 (2/2/0)
(B)	4	0	6 (1/0/0)	7 (1/1/0)
(d_{16},B)	4	0	8 (1/0/0)	10 (1/1/0)
(d_{32},B)	4	0	12 (1/0/0)	13 (1/2/0)
([B])	4	0	10 (2/0/0)	10 (2/1/0)
([B],I)	4	0	10 (2/0/0)	10 (2/1/0)
([B],d_{16})	4	0	12 (2/0/0)	13 (2/1/0)
([B],I,d_{16})	4	0	12 (2/0/0)	13 (2/1/0)
([B],d_{32})	4	0	12 (2/0/0)	14 (2/2/0)
([B],I,d_{32})	4	0	12 (2/0/0)	14 (2/2/0)
([d_{16},B])	4	0	12 (2/0/0)	13 (2/1/0)
([d_{16},B],I)	4	0	12 (2/0/0)	13 (2/1/0)
([d_{16},B],d_{16})	4	0	14 (2/0/0)	16 (2/2/0)
([d_{16},B],I,d_{16})	4	0	14 (2/0/0)	16 (2/2/0)
([d_{16},B],d_{32})	4	0	14 (2/0/0)	17 (2/2/0)
([d_{16},B],I,d_{32})	4	0	14 (2/0/0)	17 (2/2/0)
([d_{32},B])	4	0	16 (2/0/0)	17 (2/2/0)
([d_{32},B],I)	4	0	16 (2/0/0)	17 (2/2/0)
([d_{32},B],d_{16})	4	0	18 (2/0/0)	20 (2/2/0)
([d_{32},B],I,d_{16})	4	0	18 (2/0/0)	20 (2/2/0)
([d_{32},B],d_{32})	4	0	18 (2/0/0)	21 (2/3/0)
([d_{32},B],I,d_{32})	4	0	18 (2/0/0)	21 (2/3/0)

B＝ベース・アドレス；0、An、PC、Xn、An＋Xn、PC＋Xn.フォームは実行時間に影響を与えません。
I＝インデックス；0、Xn
%＝実効アドレスのフェッチではクロック・サイクルは発生しません。
注：Xnを同時にBとIに入れることはできません。Xnのスケーリングおよびサイズは実行時間に影響を与えません。

11.6.2 イミディエイト実効アドレスのフェッチ(fiea)

イミディエイト実効アドレスのフェッチの表は、プロセッサがイミディエイト・ソース・オペランドをフェッチし、指定されたデスティネーション・オペランドを計算してフェッチするために必要なクロック周期数を示しています。2ワード命令の場合、この表はプロセッサが命令の第2ワードをフェッチして、指定されたソース・オペランドまたは単一オペランドを計算してフェッチするために必要なクロック周期数を示します。実効アドレスはフォーマットで分類されています(「2.5 実効アドレス・エンコーディングの概要」参照)。命令-キャッシュ・ケースおよびノー・キャッシュ・ケースでは、合計クロック・サイクル数はかっこの外側に記入されています。リード、プリフェッチ、およびライト・サイクル数は、かっこ内に(r/p/w)の形式で記載されています。これらは合計クロック・サイクル数に含まれています。

すべての実行時間データは、2クロックのリードおよびライトを想定しています。

アドレス・モード	ヘッド	テール	Iキャッシュ・ケース	ノー・キャッシュ・ケース
単一実効アドレスの命令フォーマット				
% #⟨data⟩.W,Dn	2+op head	0	2 (0/0/0)	2 (0/1/0)
% #⟨data⟩.L,Dn	4+op head	0	4 (0/0/0)	4 (0/1/0)
#⟨data⟩.W,(An)	1	1	3 (1/0/0)	4 (1/1/0)
#⟨data⟩.L,(An)	1	0	4 (1/0/0)	5 (1/1/0)
#⟨data⟩.W,(An)+	2	1	5 (1/0/0)	5 (1/1/0)
#⟨data⟩.L,(An)+	4	1	7 (1/0/0)	7 (1/1/0)
#⟨data⟩.W,−(An)	2	2	4 (1/0/0)	4 (1/1/0)
#⟨data⟩.L,−(An)	2	0	4 (1/0/0)	5 (1/1/0)
#⟨data⟩.W,(d_{16},An)	2	0	4 (1/0/0)	5 (1/1/0)
#⟨data⟩.L,(d_{16},An)	4	0	6 (1/0/0)	8 (1/2/0)
#⟨data⟩.W,$XXX.W	4	2	6 (1/0/0)	6 (1/1/0)
#⟨data⟩.L,$XXX.W	6	2	8 (1/0/0)	8 (1/2/0)
#⟨data⟩.W,$XXX.L	3	0	6 (1/0/0)	7 (1/2/0)
#⟨data⟩.L,$XXX.L	5	0	8 (1/0/0)	9 (1/2/0)
#⟨data⟩.W,#⟨data⟩.L	6+op head	0	6 (0/0/0)	6 (0/2/0)
簡潔フォーマット拡張ワード				
#⟨data⟩.W,(d_8,An,Xn) or (d_8,PC,Xn)	6	2	8 (1/0/0)	8 (1/2/0)
#⟨data⟩.L,(d_8,An,Xn) or (d_8,PC,Xn)	8	2	10 (1/0/0)	10 (1/2/0)
全フォーマット拡張ワード				
#⟨data⟩.W,(d_{16},An) or (d_{16},PC)	4	0	8 (1/0/0)	9 (1/2/0)
#⟨data⟩.L,(d_{16},An) or (d_{16},PC)	6	0	10 (1/0/0)	11 (1/2/0)
#⟨data⟩.W,(d_{16},An,Xn) or (d_{16},PC,Xn)	6	0	8 (1/0/0)	9 (1/2/0)
#⟨data⟩.L,(d_{16},An,Xn) or (d_{16},PC,Xn)	8	0	10 (1/0/0)	11 (1/2/0)
#⟨data⟩.W,([d_{16},An]) or ([d_{16},PC])	4	0	12 (2/0/0)	12 (2/2/0)
#⟨data⟩.L,([d_{16},An]) or ([d_{16},PC])	6	0	14 (2/0/0)	14 (2/2/0)
#⟨data⟩.W,([d_{16},An],Xn) or ([d_{16},PC],Xn)	4	0	12 (2/0/0)	12 (2/2/0)
#⟨data⟩.L,([d_{16},An],Xn) or ([d_{16},PC],Xn)	6	0	14 (2/0/0)	14 (2/2/0)
#⟨data⟩.W,([d_{16},An],d_{16}) or ([d_{16},PC],d_{16})	4	0	14 (2/0/0)	15 (2/2/0)
#⟨data⟩.L,([d_{16},An],d_{16}) or ([d_{16},PC],d_{16})	6	0	16 (2/0/0)	17 (2/3/0)
#⟨data⟩.W,([d_{16},An],Xn,d_{16}) or ([d_{16},PC],Xn,d_{16})	4	0	14 (2/0/0)	15 (2/2/0)
#⟨data⟩.L,([d_{16},An],Xn,d_{16}) or ([d_{16},PC],Xn,d_{16})	6	0	16 (2/0/0)	17 (2/3/0)
#⟨data⟩.W,([d_{16},An],d_{32}) or ([d_{16},PC],d_{32})	4	0	14 (2/0/0)	16 (2/3/0)
#⟨data⟩.L,([d_{16},An],d_{32}) or ([d_{16},PC],d_{32})	6	0	16 (2/0/0)	18 (2/3/0)
#⟨data⟩.W,([d_{16},An],Xn,d_{32}) or ([d_{16},PC],Xn,d_{32})	4	0	14 (2/0/0)	16 (2/3/0)

アドレス・モード	ヘッド	テール	Iキャッシュ・ケース	ノー・キャッシュ・ケース
全フォーマット拡張ワード（つづき）				
#<data>.L,([d_{16},An],Xn,d_{32}) or ([d_{16},PC],Xn,d_{32})	6	0	16 (2/0/0)	18 (2/3/0)
#<data>.W,(B)	6	0	8 (1/0/0)	9 (1/1/0)
#<data>.L,(B)	8	0	10 (1/0/0)	11 (1/2/0)
#<data>.W,(d_{16},B)	6	0	10 (1/0/0)	12 (1/2/0)
#<data>.L,(d_{16},B)	8	0	12 (1/0/0)	14 (1/2/0)
#<data>.W,(d_{32},B)	10	0	14 (1/0/0)	16 (1/2/0)
#<data>.L,(d_{32},B)	12	0	16 (1/0/0)	18 (1/3/0)
#<data>.W,([B])	6	0	12 (2/0/0)	12 (2/1/0)
#<data>.L,([B])	8	0	14 (2/0/0)	14 (2/2/0)
#<data>.W,([B],I)	6	0	12 (2/0/0)	12 (2/1/0)
#<data>.L,([B],I)	8	0	14 (2/0/0)	14 (2/2/0)
#<data>.W,([B],d_{16})	6	0	14 (2/0/0)	15 (2/2/0)
#<data>.L,([B],d_{16})	8	0	16 (2/0/0)	17 (2/2/0)
#<data>.W,([B],I,d_{16})	6	0	14 (2/0/0)	15 (2/2/0)
#<data>.L,([B],I,d_{16})	8	0	16 (2/0/0)	17 (2/2/0)
#<data>.W,([B],d_{32})	6	0	14 (2/0/0)	16 (2/2/0)
#<data>.L,([B],d_{32})	8	0	16 (2/0/0)	18 (2/3/0)
#<data>.W,([B],I,d_{32})	6	0	14 (2/0/0)	16 (2/2/0)
#<data>.L,([B],I,d_{32})	8	0	16 (2/0/0)	18 (2/3/0)
#<data>.W,([d_{16},B])	6	0	14 (2/0/0)	15 (2/2/0)
#<data>.L,([d_{16},B])	8	0	16 (2/0/0)	17 (2/2/0)
#<data>.W,([d_{16},B],I)	6	0	14 (2/0/0)	15 (2/2/0)
#<data>.L,([d_{16},B],I)	8	0	16 (2/0/0)	17 (2/2/0)
#<data>.W,([d_{16},B],d_{16})	6	0	16 (2/0/0)	18 (2/2/0)
#<data>.L,([d_{16},B],d_{16})	8	0	18 (2/0/0)	20 (2/3/0)
#<data>.W,([d_{16},B],I,d_{16})	6	0	16 (2/0/0)	18 (2/2/0)
#<data>.L,([d_{16},B],I,d_{16})	8	0	18 (2/0/0)	20 (2/3/0)
#<data>.W,([d_{16},B],d_{32})	6	0	16 (2/0/0)	19 (2/3/0)
#<data>.L,([d_{16},B],d_{32})	8	0	18 (2/0/0)	21 (2/3/0)
#<data>.W,([d_{16},B],I,d_{32})	6	0	16 (2/0/0)	19 (2/3/0)
#<data>.L,([d_{16},B],I,d_{32})	8	0	18 (2/0/0)	21 (2/3/0)
#<data>.W,([d_{32},B])	6	0	18 (2/0/0)	19 (2/2/0)
#<data>.L,([d_{32},B])	8	0	20 (2/0/0)	21 (2/3/0)
#<data>.W,([d_{32},B],I)	6	0	18 (2/0/0)	19 (2/2/0)
#<data>.L,([d_{32},B],I)	8	0	20 (2/0/0)	21 (2/3/0)
#<data>.W,([d_{32},B],d_{16})	6	0	20 (2/0/0)	22 (2/3/0)
#<data>.L,([d_{32},B],d_{16})	8	0	22 (2/0/0)	24 (2/3/0)
#<data>.W,([d_{32},B],I,d_{16})	6	0	20 (2/0/0)	22 (2/3/0)
#<data>.L,([d_{32},B],I,d_{16})	8	0	22 (2/0/0)	24 (2/3/0)
#<data>.W,([d_{32},B],d_{32})	6	0	20 (2/0/0)	23 (2/3/0)
#<data>.L,([d_{32},B],d_{32})	8	0	22 (2/0/0)	25 (2/4/0)
#<data>.W,([d_{32},B],I,d_{32})	6	0	20 (2/0/0)	23 (2/3/0)
#<data>.L,([d_{32},B],I,d_{32})	8	0	22 (2/0/0)	25 (2/4/0)

B＝ベース・アドレス；0、An、PC、Xn、An＋Xn、PC＋Xn.フォームは実行時間に影響を与えません。
I＝インデックス；0、Xn
％＝イミディエイト実効アドレスのフェッチの全ヘッド実行時間には、その操作のヘッド時間が含まれます。
注：Xnを同時にBとIに入れることはできません。Xnのスケーリングおよびサイズは実行時間に影響を与えません。

11.6.3　実効アドレスの計算(cea)

実効アドレス計算の表は、プロセッサが指定された実効アドレスを計算するのに必要なクロック周期数を示しています。フェッチ時間は、メモリ間接アドレッシング・モードの第1レベルの間接アドレッシングに対する時間だけを含んでいます。実効アドレスはフォーマットで分類されています

(「2.5　実効アドレス・エンコーディングの概要」参照)。命令-キャッシュ・ケースおよびノー・キャッシュ・ケースでは、合計クロック・サイクル数はかっこの外側に記入されています。リード、プリフェッチ、およびライト・サイクル数は、かっこ内に(r/p/w)の形式で記載されています。これらは合計クロック・サイクル数に含まれています。

すべての実行時間データは、2クロックのリードおよびライトを想定しています。

	アドレス・モード	ヘッド	テール	Iキャッシュ・ケース	ノー・キャッシュ・ケース
単一実効アドレス命令のフォーマット					
%	Dn	—	—	0 (0/0/0)	0 (0/0/0)
%	An	—	—	0 (0/0/0)	0 (0/0/0)
	(An)	2+op head	0	2 (0/0/0)	2 (0/0/0)
	(An)+	0	0	2 (0/0/0)	2 (0/0/0)
	−(An)	2+op head	0	2 (0/0/0)	2 (0/0/0)
	(d_{16},An) or (d_{16},PC)	2+op head	0	2 (0/0/0)	2 (0/1/0)
	(xxx).W	2+op head	0	2 (0/0/0)	2 (0/1/0)
	(xxx).L	4+op head	0	4 (0/0/0)	4 (0/1/0)
簡潔フォーマット拡張ワード					
	(d_8,An,Xn) or (d_8,PC,Xn)	4+op head	0	4 (0/0/0)	4 (0/1/0)
全フォーマット拡張ワード					
	(d_{16},An) or (d_{16},PC)	2	0	6 (0/0/0)	6 (0/1/0)
	(d_{16},An,Xn) or (d_{16},PC,Xn)	6+op head	0	6 (0/0/0)	6 (0/1/0)
	([d_{16},An]) or ([d_{16},PC])	2	0	10 (1/0/0)	10 (1/1/0)
	([d_{16},An],Xn) or ([d_{16},PC],Xn)	2	0	10 (1/0/0)	10 (1/1/0)
	([d_{16},An],d_{16}) or ([d_{16},PC],d_{16})	2	0	12 (1/0/0)	13 (1/2/0)
	([d_{16},An],Xn,d_{16}) or ([d_{16},PC],Xn,d_{16})	2	0	12 (1/0/0)	13 (1/2/0)
	([d_{16},An],d_{32}) or ([d_{16},PC],d_{32})	2	0	12 (1/0/0)	13 (1/2/0)
	([d_{16},An],Xn,d_{32}) or ([d_{16},PC],Xn,d_{32})	2	0	12 (1/0/0)	13 (1/2/0)
	(B)	6+op head	0	6 (0/0/0)	6 (0/1/0)
	(d_{16},B)	4	0	8 (0/0/0)	9 (0/1/0)
	(d_{32},B)	4	0	12 (0/0/0)	12 (0/2/0)
	([B])	4	0	10 (1/0/0)	10 (1/1/0)
	([B],I)	4	0	10 (1/0/0)	10 (1/1/0)
	([B],d_{16})	4	0	12 (1/0/0)	13 (1/1/0)
	([B],I,d_{16})	4	0	12 (1/0/0)	13 (1/1/0)
	([B],d_{32})	4	0	12 (1/0/0)	13 (1/2/0)
	([B],I,d_{32})	4	0	12 (2/0/0)	13 (1/2/0)
	([d_{16},B])	4	0	12 (1/0/0)	13 (1/1/0)
	([d_{16},B],I)	4	0	12 (1/0/0)	13 (1/1/0)
	([d_{16},B],d_{16})	4	0	14 (1/0/0)	16 (1/2/0)
	([d_{16},B],I,d_{16})	4	0	14 (1/0/0)	16 (1/2/0)
	([d_{16},B],d_{32})	4	0	14 (1/0/0)	16 (1/2/0)
	([d_{16},B],I,d_{32})	4	0	14 (1/0/0)	16 (1/2/0)
	([d_{32},B])	4	0	16 (1/0/0)	17 (1/2/0)
	([d_{32},B],I)	4	0	16 (1/0/0)	17 (1/2/0)
	([d_{32},B],d_{16})	4	0	18 (1/0/0)	20 (1/2/0)
	([d_{32},B],I,d_{16})	4	0	18 (1/0/0)	20 (1/2/0)
	([d_{32},B],d_{32})	4	0	18 (1/0/0)	20 (1/3/0)
	([d_{32},B],I,d_{32})	4	0	18 (1/0/0)	20 (1/3/0)

B＝ベース・アドレス；0、An、PC、Xn、An＋Xn、PC＋Xn.フォームは実行時間に影響を与えません。
I＝インデックス；0、Xn
%＝実効アドレスのフェッチではクロック・サイクルは発生しません。
注：Xnを同時にBとIに入れることはできません。Xnのスケーリングおよびサイズは実行時間に影響を与えません。

11. 6. 4　イミディエイト実効アドレス計算モード(ciea)

イミディエイト実効アドレス計算の表は、プロセッサがイミディエイト・ソース・オペランドをフェッチし、指定されたデスティネーション実効アドレスを計算するのに必要なクロック周期数を示しています。2ワード命令の場合、この表はプロセッサが命令の第2ワードをフェッチして、指定されたソース・オペランドまたは単一オペランドを計算するために必要なクロック周期数を示します。フェッチ時間は、メモリ間接アドレッシング・モードの第1レベルの間接アドレッシングに対する時間だけを含んでいます。実効アドレスはフォーマットで分類されています(「2. 5　実効アドレス・エンコーディングの概要」参照)。命令－キャッシュ・ケースおよびノー・キャッシュ・ケースでは、合計クロック・サイクル数はかっこの外側に記入されています。リード、プリフェッチ、およびライト・サイクル数は、かっこ内に(r/p/w)の形式で記載されています。これらは合計クロック・サイクル数に含まれています。

すべての実行時間データは、2クロックのリードおよびライトを想定しています。

	アドレス・モード	ヘッド	テール	Iキャッシュ・ケース	ノー・キャッシュ・ケース
	単一実効アドレス命令のフォーマット				
%	#(data).W,Dn	2+op head	0	2 (0/0/0)	2 (0/1/0)
%	#(data).L,Dn	4+op head	0	4 (0/0/0)	4 (0/1/0)
%	#(data).W,(An)	2+op head	0	2 (0/0/0)	2 (0/1/0)
%	#(data).L,(An)	4+op head	0	4 (0/0/0)	4 (0/1/0)
	#(data).W,(An)+	2	0	4 (0/0/0)	4 (0/1/0)
	#(data).L,(An)+	4	0	6 (0/0/0)	6 (0/1/0)
%	#(data).W,−(An)	2+op head	0	2 (0/0/0)	2 (0/1/0)
%	#(data).L,−(An)	4+op head	0	4 (0/0/0)	4 (0/1/0)
%	#(data).W,(d_{16},An)	4+op head	0	4 (0/0/0)	4 (0/1/0)
%	#(data).L,(d_{16},An)	6+op head	0	6 (0/0/0)	7 (0/2/0)
%	#(data).W,$XXX.W	4+op head	0	4 (0/0/0)	4 (0/1/0)
%	#(data).L,$XXX.W	6+op head	0	6 (0/0/0)	6 (0/2/0)
%	#(data).W,$XXX.L	6+op head	0	6 (0/0/0)	6 (0/2/0)
%	#(data).L,$XXX.L	8+op head	0	8 (0/0/0)	8 (0/2/0)
	簡潔フォーマット拡張ワード				
%	#(data).W,(d_8,An,Xn) or (d_8,PC,Xn)	6+op head	0	6 (0/0/0)	6 (0/2/0)
%	#(data).L,(d_8,An,Xn) or (d_8,PC,Xn)	8+op head	0	8 (0/0/0)	8 (0/2/0)
	全フォーマット拡張ワード				
	#(data).W,(d_{16},An) or (d_{16},PC)	4	0	8 (0/0/0)	8 (0/2/0)
	#(data).L,(d_{16},An) or (d_{16},PC)	6	0	10 (0/0/0)	10 (0/2/0)
%	#(data).W,(d_{16},An,Xn) or (d_{16},PC,Xn)	8+op head	0	8 (0/0/0)	8 (0/2/0)
%	#(data).L,(d_{16},An,Xn) or (d_{16},PC,Xn)	10+op head	0	10 (0/0/0)	10 (0/2/0)
	#(data).W,([d_{16},An]) or ([d_{16},PC])	4	0	12 (1/0/0)	12 (1/2/0)
	#(data).L,([d_{16},An]) or ([d_{16},PC])	6	0	14 (1/0/0)	14 (1/1/0)
	#(data).W,([d_{16},An],Xn) or ([d_{16},PC],Xn)	4	0	12 (1/0/0)	12 (1/2/0)
	#(data).L,([d_{16},An],Xn) or ([d_{16},PC],Xn)	6	0	14 (1/0/0)	14 (1/1/0)
	#(data).W,([d_{16},An],d_{16}) or ([d_{16},PC],d_{16})	4	0	14 (1/0/0)	15 (1/2/0)
	#(data).L,([d_{16},An],d_{16}) or ([d_{16},PC],d_{16})	6	0	16 (1/0/0)	17 (1/3/0)
	#(data).W,([d_{16},An],Xn,d_{16}) or ([d_{16},PC],Xn,d_{16})	4	0	14 (1/0/0)	15 (1/2/0)
	#(data).L,([d_{16},An],Xn,d_{16}) or ([d_{16},PC],Xn,d_{16})	6	0	16 (1/0/0)	17 (1/3/0)
	#(data).W,([d_{16},An],d_{32}) or ([d_{16},PC],d_{32})	4	0	14 (1/0/0)	16 (1/3/0)
	#(data).L,([d_{16},An],d_{32}) or ([d_{16},PC],d_{32})	6	0	16 (1/0/0)	17 (1/3/0)
	#(data).W,([d_{16},An],Xn,d_{32}) or ([d_{16},PC],Xn,d_{32})	4	0	14 (1/0/0)	15 (1/3/0)

	アドレス・モード	ヘッド	テール	Iキャッシュ・ケース	ノー・キャッシュ・ケース
	全フォーマット拡張ワード（つづき）				
	#⟨data⟩.L,([d_{16},An],Xn,d_{32}) or ([d_{16},PC],Xn,d_{32})	6	0	16 (1/0/0)	17 (1/3/0)
%	#⟨data⟩.W,(B)	8＋op head	0	8 (0/0/0)	8 (0/1/0)
%	#⟨data⟩.L,(B)	10＋op head	0	10 (0/0/0)	10 (0/2/0)
	#⟨data⟩.W,(d_{16},B)	6	0	10 (0/0/0)	11 (0/2/0)
	#⟨data⟩.L,(d_{16},B)	8	0	12 (0/0/0)	13 (0/2/0)
	#⟨data⟩.W,(d_{32},B)	6	0	14 (0/0/0)	15 (0/2/0)
	#⟨data⟩.L,(d_{32},B)	8	0	16 (0/0/0)	17 (0/3/0)
	#⟨data⟩.W,([B])	6	0	12 (1/0/0)	12 (1/1/0)
	#⟨data⟩.L,([B])	8	0	14 (1/0/0)	14 (1/2/0)
	#⟨data⟩.W,([B],I)	6	0	12 (1/0/0)	12 (1/1/0)
	#⟨data⟩.L,([B],I)	8	0	14 (1/0/0)	14 (1/2/0)
	#⟨data⟩.W,([B],d_{16})	6	0	14 (1/0/0)	15 (1/2/0)
	#⟨data⟩.L,([B],d_{16})	8	0	16 (1/0/0)	17 (1/2/0)
	#⟨data⟩.W,([B],I,d_{16})	6	0	14 (1/0/0)	15 (1/2/0)
	#⟨data⟩.L,([B],I,d_{16})	8	0	16 (2/0/0)	17 (1/2/0)
	#⟨data⟩.W,([B],d_{32})	6	0	14 (1/0/0)	15 (1/2/0)
	#⟨data⟩.L,([B],d_{32})	8	0	16 (1/0/0)	17 (1/3/0)
	#⟨data⟩.W,([B],I,d_{32})	6	0	14 (1/0/0)	15 (1/2/0)
	#⟨data⟩.L,([B],I,d_{32})	8	0	16 (1/0/0)	17 (1/3/0)
	#⟨data⟩.W,([d_{16},B])	6	0	14 (1/0/0)	15 (1/2/0)
	#⟨data⟩.L,([d_{16},B])	8	0	16 (1/0/0)	17 (1/2/0)
	#⟨data⟩.W,([d_{16},B],I)	6	0	14 (1/0/0)	15 (1/2/0)
	#⟨data⟩.L,([d_{16},B],I)	8	0	16 (1/0/0)	17 (1/2/0)
	#⟨data⟩.W,([d_{16},B],d_{16})	6	0	16 (1/0/0)	18 (1/2/0)
	#⟨data⟩.L,([d_{16},B],d_{16})	8	0	18 (1/0/0)	20 (1/3/0)
	#⟨data⟩.W,([d_{16},B],I,d_{16})	6	0	16 (1/0/0)	18 (1/2/0)
	#⟨data⟩.L,([d_{16},B],I,d_{16})	8	0	18 (1/0/0)	20 (1/3/0)
	#⟨data⟩.W,([d_{16},B],d_{32})	6	0	16 (1/0/0)	18 (1/3/0)
	#⟨data⟩.L,([d_{16},B],d_{32})	8	0	18 (1/0/0)	20 (1/3/0)
	#⟨data⟩.W,([d_{16},B],I,d_{32})	6	0	16 (1/0/0)	18 (1/3/0)
	#⟨data⟩.L,([d_{16},B],I,d_{32})	8	0	18 (1/0/0)	20 (1/3/0)
	#⟨data⟩.W,([d_{32},B])	6	0	18 (1/0/0)	19 (1/2/0)
	#⟨data⟩.L,([d_{32},B])	8	0	20 (1/0/0)	21 (1/3/0)
	#⟨data⟩.W,([d_{32},B],I)	6	0	18 (1/0/0)	19 (1/2/0)
	#⟨data⟩.L,([d_{32},B],I)	8	0	20 (1/0/0)	21 (1/3/0)
	#⟨data⟩.W,([d_{32},B],d_{16})	6	0	20 (1/0/0)	22 (1/3/0)
	#⟨data⟩.L,([d_{32},B],d_{16})	8	0	22 (1/0/0)	24 (1/3/0)
	#⟨data⟩.W,([d_{32},B],I,d_{16})	6	0	20 (1/0/0)	22 (1/3/0)
	#⟨data⟩.L,([d_{32},B],I,d_{16})	8	0	22 (1/0/0)	24 (1/3/0)
	#⟨data⟩.W,([d_{32},B],d_{32})	6	0	20 (1/0/0)	22 (1/3/0)
	#⟨data⟩.L,([d_{32},B],d_{32})	8	0	22 (1/0/0)	24 (1/4/0)
	#⟨data⟩.W,([d_{32},B],I,d_{32})	6	0	20 (1/0/0)	22 (1/3/0)
	#⟨data⟩.L,([d_{32},B],I,d_{32})	8	0	22 (1/0/0)	24 (1/4/0)

B＝ベース・アドレス；0、An、PC、Xn、An＋Xn、PC＋Xn.フォームは実行時間に影響を与えません。
I＝インデックス；0、Xn
%＝アドレスの全ヘッド実行時間には、その操作のヘッド時間が含まれます。
注：Xnを同時にBとIに入れること5できません。Xnのスケーリングおよびサイズは実行時間に影響を与えません。

11.6.5　ジャンプ実効アドレス・モード

ジャンプ実効アドレス計算の表は、プロセッサがJMPまたはJSR命令で指定された実効アドレスを計算するのに必要なクロック周期数を示しています。フェッチ時間は、メモリ間接アドレッシング・モードでは、第1レベルの間接アドレッシングに対する時間だけを含んでいます。実効アドレスはフォーマットで分類されています(「2.5　実効アドレス・エンコーディングの概要」参照)。命令-キャッシュ・ケースおよびノー・キャッシュ・ケースでは、合計クロック・サイクル数はかっこの外側に記入されています。リード、プリフェッチ、およびライト・サイクル数は、かっこ内に(r/p/w)の形式で記載されています。これらは合計クロック・サイクル数に含まれています。

すべての実行時間データは、2クロックのリードおよびライトを想定しています。

	アドレス・モード	ヘッド	テール	Iキャッシュ・ケース	ノー・キャッシュ・ケース
単一実効アドレス命令のフォーマット					
%	(An)	2+op head	0	2 (0/0/0)	2 (0/0/0)
%	(d_{16},An)	4+op head	0	4 (0/0/0)	4 (0/0/0)
%	(xxx).W	2+op head	0	2 (0/0/0)	2 (0/0/0)
%	(xxx).L	2+op head	0	2 (0/0/0)	2 (0/0/0)
簡潔フォーマット拡張ワード					
%	(d_8,An,Xn) or (d_8,PC,Xn)	6+op head	0	6 (0/0/0)	6 (0/0/0)
全フォーマット拡張ワード					
	(d_{16},An) or (d_{16},PC)	2	0	6 (0/0/0)	6 (0/0/0)
%	(d_{16},An,Xn) or (d_{16},PC,Xn)	6+op head	0	6 (0/0/0)	6 (0/0/0)
	([d_{16},An]) or ([d_{16},PC])	2	0	10 (1/0/0)	10 (1/1/0)
	([d_{16},An],Xn) or ([d_{16},PC],Xn)	2	0	10 (1/0/0)	10 (1/1/0)
	([d_{16},An],d_{16}) or ([d_{16},PC],d_{16})	2	0	12 (1/0/0)	12 (1/1/0)
	([d_{16},An],Xn,d_{16}) or ([d_{16},PC],Xn,d_{16})	2	0	12 (1/0/0)	12 (1/1/0)
	([d_{16},An],d_{32}) or ([d_{16},PC],d_{32})	2	0	12 (1/0/0)	12 (1/1/0)
	([d_{16},An],Xn,d_{32}) or ([d_{16},PC],Xn,d_{32})	2	0	12 (1/0/0)	12 (1/1/0)
%	(B)	6+op head	0	6 (0/0/0)	6 (0/0/0)
	(d_{16},B)	4	0	8 (0/0/0)	9 (0/1/0)
	(d_{32},B)	4	0	12 (0/0/0)	13 (0/1/0)
	([B])	4	0	10 (1/0/0)	10 (1/1/0)
	([B],I)	4	0	10 (1/0/0)	10 (1/1/0)
	([B],d_{16})	4	0	12 (1/0/0)	12 (1/1/0)
	([B],I,d_{16})	4	0	12 (1/0/0)	12 (1/1/0)
	([B],d_{32})	4	0	12 (1/0/0)	12 (1/1/0)
	([B],d_{32})	4	0	12 (1/0/0)	12 (1/1/0)
	([B],I,d_{32})	4	0	12 (1/0/0)	12 (1/1/0)
	([d_{16},B])	4	0	12 (1/0/0)	13 (1/1/0)
	([d_{16},B],I)	4	0	12 (1/0/0)	13 (1/1/0)
	([d_{16},B],d_{16})	4	0	14 (1/0/0)	15 (1/1/0)
	([d_{16},B],I,d_{16})	4	0	14 (1/0/0)	15 (1/1/0)
	([d_{16},B],d_{32})	4	0	14 (1/0/0)	15 (1/1/0)
	([d_{16},B],I,d_{32})	4	0	14 (1/0/0)	15 (1/1/0)
	([d_{32},B])	4	0	16 (1/0/0)	17 (1/2/0)
	([d_{32},B],I)	4	0	16 (1/0/0)	17 (1/2/0)
	([d_{32},B],d_{16})	4	0	18 (1/0/0)	19 (1/2/0)
	([d_{32},B],I,d_{16})	4	0	18 (1/0/0)	19 (1/2/0)
	([d_{32},B],d_{32})	4	0	18 (1/0/0)	19 (1/2/0)
	([d_{32},B],I,d_{32})	4	0	18 (1/0/0)	19 (1/2/0)

B＝ベース・アドレス；0、An、PC、Xn、An＋Xn、PC＋Xn.フォームは実行時間に影響を与えません。
I＝インデックス；0、Xn
%＝アドレスの全ヘッド実行時間には、その操作のヘッド時間が含まれます。
注：Xnを同時にBとIに入れることはできません。Xnのスケーリングおよびサイズは実行時間に影響を与えません。

11. 6. 6 MOVE命令

MOVE命令実行時間の表は、プロセッサが指定されたソースおよびデスティネーションの実効アドレスを計算して、メモリ間接アドレッシング・モードの第1レベルの間接アドレッシングを含め、MOVEまたはMOVEA命令を実行するのに必要なクロック周期数を示しています。ほとんどのMOVE操作で、フェッチ実効アドレス表が必要です(ソース、デスティネーションに依存)。デスティネーションの実効アドレスはフォーマットで分類されています(「2. 5 実効アドレス・エンコーディングの概要」参照)。命令キャッシュ・ケースおよびノー・キャッシュ・ケースでは、合計クロック・サイクル数はかっこの外側に記入されています。リード、プリフェッチ、およびライト・サイクル数は、かっこ内に(r/p/w)の形式で記載されています。これらは合計クロック・サイクル数に含まれています。

すべての実行時間データは、2クロックのリードおよびライトを想定しています。

	MOVEのソース、デスティネーション	ヘッド	テール	Iキャッシュ・ケース	ノー・キャッシュ・ケース
単一実効アドレス命令のフォーマット					
	MOVE Rn, Dn	2	0	2 (0/0/0)	2 (0/1/0)
	MOVE Rn, An	2	0	2 (0/0/0)	2 (0/1/0)
*	MOVE EA,An	0	0	2 (0/0/0)	2 (0/1/0)
*	MOVE EA,Dn	0	0	2 (0/0/0)	2 (0/1/0)
	MOVE Rn,(An)	0	1	3 (0/0/1)	4 (0/1/1)
*	MOVE SOURCE, (An)	2	0	4 (0/0/1)	5 (0/1/1)
	MOVE Rn,(An)+	0	1	3 (0/0/1)	4 (0/1/1)
*	MOVE SOURCE, (An)+	2	0	4 (0/0/1)	5 (0/1/1)
	MOVE Rn, −(An)	0	2	4 (0/0/1)	4 (0/1/1)
*	MOVE SOURCE, −(An)	2	0	4 (0/0/1)	5 (0/1/1)
*	MOVE EA, (d_{16},An)	2	0	4 (0/0/1)	5 (0/1/1)
*	MOVE EA,XXX.W	2	0	4 (0/0/1)	5 (0/1/1)
*	MOVE EA,XXX.L	0	0	6 (0/0/1)	7 (0/2/1)
簡潔フォーマット拡張ワード					
*	MOVE EA, (d_8,An,Xn)	4	0	6 (0/0/1)	7 (0/1/1)
全フォーマット拡張ワード					
*	MOVE EA, (d_{16},An) or (d_{16},PC)	2	0	8 (0/0/1)	9 (0/2/1)
*	MOVE EA, (d_{16},An,Xn) or (d_{16},PC,Xn)	2	0	8 (0/0/1)	9 (0/2/1)
*	MOVE EA, ([d_{16},An],Xn) or ([d_{16},PC],Xn)	2	0	10 (1/0/1)	11 (1/2/1)
*	MOVE EA,([d_{16},An],d_{16}) or ([d_{16},PC],d_{16})	2	0	12 (1/0/1)	14 (1/2/1)
*	MOVE EA,([d_{16},An],Xn,d_{16}) or ([d_{16},PC],Xn,d_{16})	2	0	12 (1/0/1)	14 (1/2/1)
*	MOVE EA,([d_{16},An],d_{32}) or ([d_{16},PC],d_{32})	2	0	14 (1/0/1)	16 (1/3/1)
*	MOVE EA,([d_{16},An],Xn,d_{32}) or ([d_{16},PC],Xn,d_{32})	2	0	14 (1/0/1)	16 (1/3/1)
*	MOVE EA,(B)	4	0	8 (0/0/1)	9 (0/1/1)
*	MOVE EA,(d_{16},B)	4	0	10 (0/0/1)	12 (0/2/1)
*	MOVE EA,(d_{32},B)	4	0	14 (0/0/1)	16 (0/2/1)
*	MOVE EA,([B])	4	0	10 (1/0/1)	11 (1/1/1)
*	MOVE EA,([B],I)	4	0	10 (1/0/1)	11 (1/1/1)
*	MOVE EA,([B],d_{16})	4	0	12 (1/0/1)	14 (1/2/1)
*	MOVE EA,([B],I,d_{16})	4	0	12 (1/0/1)	14 (1/2/1)
*	MOVE EA,([B],d_{32})	4	0	14 (1/0/1)	16 (1/2/1)
*	MOVE EA,([B],I,d_{32})	4	0	14 (1/0/1)	16 (1/2/1)
*	MOVE EA,([d_{16},B])	4	0	12 (1/0/1)	14 (1/2/1)
*	MOVE EA,([d_{16},B],I)	4	0	12 (1/0/1)	14 (1/2/1)
*	MOVE EA,([d_{16},B],d_{16})	4	0	14 (1/0/1)	17 (1/2/1)
*	MOVE EA,([d_{16},B],I,d_{16})	4	0	14 (1/0/1)	17 (1/2/1)
*	MOVE EA,([d_{16},B],d_{32})	4	0	16 (1/0/1)	19 (1/3/1)

MOVEのソース、デスティネーション	ヘッド	テール	Iキャッシュ・ケース	ノー・キャッシュ・ケース

全フォーマット拡張ワード（つづき）

	MOVEのソース、デスティネーション	ヘッド	テール	Iキャッシュ・ケース	ノー・キャッシュ・ケース
*	MOVE EA,([d_{16},B],I,d_{32})	4	0	16 (1/0/1)	19 (1/3/1)
*	MOVE EA,([d_{32},B])	4	0	16 (1/0/1)	18 (1/2/1)
*	MOVE EA,([d_{32},B],I)	4	0	16 (1/0/1)	18 (1/2/1)
*	MOVE EA,([d_{32},B],d_{16})	4	0	18 (1/0/1)	21 (1/3/1)
*	MOVE EA,([d_{32},B],I,d_{16})	4	0	18 (1/0/1)	21 (1/3/1)
*	MOVE EA,([d_{32},B],d_{32})	4	0	20 (1/0/1)	23 (1/3/1)
*	MOVE EA,([d_{32},B],I,d_{32})	4	0	20 (1/0/1)	23 (1/3/1)

*　実効アドレスのフェッチ時間を加算します。
Rn　データ・レジスタまたはアドレス・レジスタ
SOURCE　メモリまたはイミディエイト・データ・アドレス・モード
EA　実効アドレス

11.6.7　特殊目的のMOVE命令

特殊目的MOVE命令実行時間の表は、プロセッサが制御レジスタまたは指定された実効アドレスに対して、特殊目的のMOVE命令をフェッチし、計算し、実行するのに必要なクロック周期数を示しています。脚注に該当する実効アドレス時間を加算する場合を示します。合計クロック・サイクル数はかっこの外側に記入されています。リード、プリフェッチ、およびライト・サイクル数は、かっこ内に(r/p/w)の形式で記載されています。これらは合計クロック・サイクル数に含まれています。

すべての実行時間データは、2クロックのリードおよびライトを想定しています。

	命令		ヘッド	テール	Iキャッシュ・ケース	ノー・キャッシュ・ケース
	EXG	Ry,Rx	4	0	4 (0/0/0)	4 (0/1/0)
	MOVEC	Cr,Rn	6	0	6 (0/0/0)	6 (0/1/0)
	MOVEC	Rn,Cr－A	6	0	6 (0/0/0)	6 (0/1/0)
	MOVEC	Rn,Cr－B	4	0	12 (0/0/0)	12 (0/1/0)
	MOVE	CCR,Dn	2	0	4 (0/0/0)	4 (0/1/0)
*	MOVE	CCR,Mem	2	0	4 (0/0/1)	5 (0/1/1)
	MOVE	Dn,CCR	4	0	4 (0/0/0)	4 (0/1/0)
*	MOVE	EA,CCR	0	0	4 (0/0/0)	4 (0/1/0)
	MOVE	SR,Dn	2	0	4 (0/0/0)	4 (0/1/0)
*	MOVE	SR,Mem	2	0	4 (0/0/1)	5 (0/1/1)
#	MOVE	EA,SR	0	0	8 (0/0/0)	10 (0/2/0)
% +	MOVEM	EA,RL	2	0	8＋4n (n/0/0)	8＋4n (n/1/0)
% +	MOVEM	RL,EA	2	0	4＋2n (0/0/n)	4＋2n (0/1/n)
	MOVEP.W	Dn,(d_{16},An)	4	0	10 (0/0/2)	10 (0/1/2)
	MOVEP.W	(d_{16},An),Dn	2	0	10 (2/0/0)	10 (2/1/0)
	MOVEP.L	Dn,(d_{16},An)	4	0	14 (0/0/4)	14 (0/1/4)
	MOVEP.L	(d_{16},An),Dn	2	0	14 (4/0/0)	14 (4/1/0)
%	MOVES	EA,Rn	3	0	7 (1/0/0)	7 (1/1/0)
%	MOVES	Rn,EA	2	1	5 (0/0/1)	6 (0/1/1)
	MOVE	USP,An	4	0	4 (0/0/0)	4 (0/1/0)
	MOVE	An,USP	4	0	4 (0/0/0)	4 (0/1/0)
	SWAP	Dn	4	0	4 (0/0/0)	4 (0/1/0)

CR－A 制御レジスタUSP、VBR、CAAR、MSP,およびISP＋
CR－A 制御レジスタSFC、DFC、およびCACR
n転送するレジスタ数 (n＞0)
RL レジスタ・リスト
*実効アドレスの計算時間を加算します。
#実効アドレスのフェッチ時間を加算します。
%イミディエイト・アドレスの計算時間を加算します。

MOVEM EA、RL ── nレジスタ(n＞0)およびwウエイト・ステートの場合
Iキャッシュ・ケース・タイミング＝w≦2：(8＋4n)
w＞2：(8＋4n)＋(w－2) n
すべてのウエイト・ステートで、テイル＝0
MOVEM RL、EA ── nレジスタ(n＞0)およびwウエイト・ステートの場合
Iキャッシュ・ケース・タイミング＝w≦2：(4＋2n)＋(n－1) w
w＞2：(4＋2n)＋(n－1) w＋(w－2)
テール＝w≦2：(n－1) w
w＞2：(n) w＋(n) (w－2)

11.6.8 算術/論理演算命令

算術/論理演算実行時間の表は、プロセッサが指定されたアドレッシング・モードを使用して、指定された算術/論理演算を実行するのに必要なクロック周期数を示しています。脚注にアドレスのフェッチまたはイミディエイト実効アドレスのフェッチ時間を加算する場合を示します。命令-キャッシュ・ケースおよびノー・キャッシュ・ケースでは、合計クロック・サイクル数はかっこの外側に記入されています。リード、プリフェッチ、およびライト・サイクル数は、かっこ内に(r/p/w)の形式で記載されています。これらは合計クロック・サイクル数に含まれています。

すべての実行時間データは、2クロックのリードおよびライトを想定しています。

	命	令	ヘッド	テール	Iキャッシュ・ケース	ノー・キャッシュ・ケース
	ADD	Rn,Dn	2	0	2 (0/0/0)	2 (0/1/0)
	ADDA.W	Rn,An	4	0	4 (0/0/0)	4 (0/1/0)
	ADDA.L	Rn,An	2	0	2 (0/0/0)	2 (0/1/0)
*	ADD	EA,Dn	0	0	2 (0/0/0)	2 (0/1/0)
*	ADD.W	EA,An	0	0	4 (0/0/0)	4 (0/1/0)
*	ADDA.L	EA,An	0	0	2 (0/0/0)	2 (0/1/0)
*	ADD	Dn,EA	0	1	3 (0/0/1)	4 (0/1/1)
	AND	Dn,Dn	2	0	2 (0/0/0)	2 (0/1/0)
*	AND	EA,Dn	0	0	2 (0/0/0)	2 (0/1/0)
*	AND	Dn,EA	0	1	3 (0/0/1)	4 (0/1/1)
	EOR	Dn,Dn	2	0	2 (0/0/0)	2 (0/1/0)
*	EOR	Dn,EA	0	1	3 (0/0/1)	4 (0/1/1)
	OR	Dn,Dn	2	0	2 (0/0/0)	2 (0/1/0)
*	OR	EA,Dn	0	0	2 (0/0/0)	2 (0/1/0)
*	OR	Dn,EA	0	1	3 (0/0/1)	4 (0/1/1)
	SUB	Rn,Dn	2	0	2 (0/0/0)	2 (0/1/0)
*	SUB	EA,Dn	0	0	2 (0/0/0)	2 (0/1/0)
*	SUB	Dn,EA	0	1	3 (0/0/1)	4 (0/1/1)
	SUBA.W	Rn,An	4	0	4 (0/0/0)	4 (0/1/0)
	SUBA.L	Rn,An	2	0	2 (0/0/0)	2 (0/1/0)
*	SUBA.W	EA,An	0	0	4 (0/0/0)	4 (0/1/0)
*	SUBA.L	EA,An	0	0	2 (0/0/0)	2 (0/1/0)
	CMP	Rn,Dn	2	0	2 (0/0/0)	2 (0/1/0)
*	CMP	EA,Dn	0	0	2 (0/0/0)	2 (0/1/0)
	CMPA	Rn,An	4	0	4 (0/0/0)	4 (0/1/0)
*	CMPA	EA,An	0	0	4 (0/0/0)	4 (0/1/0)
** +	CMP2	EA,Rn	2	0	20 (1/0/0)	20 (1/1/0)
* +	MULS.W	EA,Dn	2	0	28 (0/0/0)	28 (0/1/0)
** +	MULS.L	EA,Dn	2	0	44 (0/0/0)	44 (0/1/0)
* +	MULU.W	EA,Dn	2	0	28 (0/0/0)	28 (0/1/0)
** +	MULU.L	EA,Dn	2	0	44 (0/0/0)	44 (0/1/0)
+	DIVS.W	Dn,Dn	2	0	56 (0/0/0)	56 (0/1/0)
* +	DIVS.W	EA,Dn	0	0	56 (0/0/0)	56 (0/1/0)
** +	DIVS.L	Dn,Dn	6	0	90 (0/0/0)	90 (0/1/0)
** +	DIVS.L	EA,Dn	0	0	90 (0/0/0)	90 (0/1/0)
+	DIVU.W	Dn,Dn	2	0	44 (0/0/0)	44 (0/1/0)
* +	DIVU.W	EA,Dn	0	0	44 (0/0/0)	44 (0/1/0)
** +	DIVU.L	Dn,Dn	6	0	78 (0/0/0)	78 (0/1/0)
** +	DIVU.L	EA,Dn	0	0	78 (0/0/0)	78 (0/1/0)

*イミディエイト実効アドレスのフェッチ時間を加算します。
**実効アドレスのフェッチ時間を加算します。
＋ 最大時間を示します(実際の時間はデータ・タイプによります)。

11. 6. 9　イミディエイト算術/論理演算命令

イミディエイト算術/論理演算実行時間の表は、プロセッサがイミディエイトのソース・データ値をフェッチし、指定されたデスティネーション・アドレッシング・モードを使用して、指定された算術/論理演算を実行するのに必要なクロック周期数を示しています。脚注にアドレスのフェッチまたはイミディエイト実効アドレスのフェッチの時間を加算する場合を示します。命令-キャッシュ・ケースおよびノー・キャッシュ・ケースでは、合計クロック・サイクル数はかっこの外側に記入されています。リード、プリフェッチ、およびライト・サイクル数は、かっこ内に(r/p/w)の形式で記載されています。これらは合計クロック・サイクル数に含まれています。

すべての実行時間データは、2クロックのリードおよびライトを想定しています。

	命　令		ヘッド	テール	Iキャッシュ・ケース	ノー・キャッシュ・ケース
	MOVEQ	#(data),Dn	2	0	2 (0/0/0)	2 (0/1/0)
	ADDQ	#(data),Rn	2	0	2 (0/0/0)	2 (0/1/0)
*	ADDQ	#(data),Mem	0	1	3 (0/0/1)	4 (0/1/1)
	SUBQ	#(data),Rn	2	0	2 (0/0/0)	2 (0/1/0)
*	SUBQ	#(data),Mem	0	1	3 (0/0/1)	4 (0/1/1)
**	ADDI	#(data),Dn	2	0	2 (0/0/0)	2 (0/1/0)
**	ADDI	#(data),Mem	0	1	3 (0/0/1)	4 (0/1/1)
**	ANDI	#(data),Dn	2	0	2 (0/0/0)	2 (0/1/0)
**	ANDI	#(data),Mem	0	1	3 (0/0/1)	4 (0/1/1)
**	EORI	#(data),Dn	2	0	2 (0/0/0)	2 (0/1/0)
**	EORI	#(data),Mem	0	1	3 (0/0/1)	4 (0/1/1)
**	ORI	#(data),Dn	2	0	2 (0/0/0)	2 (0/1/0)
**	ORI	#(data),Mem	0	1	3 (0/0/1)	4 (0/1/1)
**	SUBI	#(data),Dn	2	0	2 (0/0/0)	2 (0/1/0)
**	SUBI	#(data),Mem	0	1	3 (0/0/1)	4 (0/1/1)
**	CMPI	#(data),Dn	2	0	2 (0/0/0)	2 (0/1/0)
**	CMPI	#(data),Mem	0	0	2 (0/0/0)	2 (0/1/0)

*　実効アドレスのフェッチ時間を加算します。
**　イミディエイト実効アドレスのフェッチ時間を加算します。

11. 6. 10　2進化10進および拡張命令

2進化10進および拡張命令実行時間の表は、プロセッサが指定されたアドレッシング・モードを使用して、指定された演算を実行するのに必要なクロック周期数を示しています。これらの命令に対する有効な合計実行時間を計算するのに、ほかの表を使用する必要はありません。命令-キャッシュ・ケースおよびノー・キャッシュ・ケースでは、合計クロック・サイクル数はかっこの外側に記入されています。リード、プリフェッチ、およびライト・サイクル数は、かっこ内に(r/p/w)の形式で記載されています。これらは合計クロック・サイクル数に含まれています。

すべての実行時間データは、2クロックのリードおよびライトを想定しています。

命　令			ヘッド	テール	Iキャッシュ・ケース	ノー・キャッシュ・ケース
	ABCD	Dn,Dn	0	0	4 (0/0/0)	4 (0/1/0)
	ABCD	−(An),−(An)	2	1	13 (2/0/1)	14 (2/1/1)
	SBCD	Dn,Dn	0	0	4 (0/0/0)	4 (0/1/0)
	SBCD	−(An),−(An)	2	1	13 (2/0/1)	14 (2/1/1)
	ADDX	Dn,Dn	2	0	2 (0/0/0)	2 (0/1/0)
	ADDX	−(An),−(An)	2	1	9 (2/0/1)	10 (2/1/1)
	SUBX	Dn,Dn	2	0	2 (0/0/0)	2 (0/1/0)
	SUBX	−(An),−(An)	2	1	9 (2/0/1)	10 (2/1/1)
	CMPM	(An)+,(An)+	0	0	8 (2/0/0)	8 (2/1/0)
	PACK	Dn,Dn,#〈data〉	6	0	6 (0/0/0)	6 (0/1/0)
	PACK	−(An),−(An),#〈data〉	2	1	11 (1/0/1)	11 (1/1/1)
	UNPK	Dn,Dn,#〈data〉	8	0	8 (0/0/0)	8 (0/1/0)
	UNPK	−(An),−(An),#〈data〉	2	1	11 (1/0/1)	11 (1/1/1)

11. 6. 11　単一オペランド命令

単一オペランド命令の実行時間の表は、プロセッサが指定されたアドレッシング・モードで、指定された操作を実行するのに必要なクロック周期の数を示しています。脚注に、適切な実効アドレス時間を加算する必要がある場合を示します。命令-キャッシュ・ケースおよびノー・キャッシュ・ケースでは、合計クロック・サイクル数はかっこの外側に記入されています。リード、プリフェッチ、およびライト・サイクル数は、かっこ内に(r/p/w)の形式で記載されています。これらは合計クロック・サイクル数に含まれています。

すべての実行時間データは、2クロックのリードおよびライトを想定しています。

命　令			ヘッド	テール	Iキャッシュ・ケース	ノー・キャッシュ・ケース
	CLR	Dn	2	0	2 (0/0/0)	2 (0/1/0)
**	CLR	Mem	0	1	3 (0/0/1)	4 (0/1/1)
	NEG	Dn	2	0	2 (0/0/0)	2 (0/1/0)
*	NEG	Mem	0	1	3 (0/0/1)	4 (0/1/1)
	NEGX	Dn	2	0	2 (0/0/0)	2 (0/1/0)
*	NEGX	Mem	0	1	3 (0/0/1)	4 (0/1/1)
	NOT	Dn	2	0	2 (0/0/0)	2 (0/1/0)
*	NOT	Mem	0	1	3 (0/0/1)	4 (0/1/1)
	EXT	Dn	4	0	4 (0/0/0)	4 (0/1/0)
	NBCD	Dn	0	0	6 (0/0/0)	6 (0/1/0)
	Scc	Dn	4	0	4 (0/0/0)	4 (0/1/0)
**	Scc	Mem	0	1	5 (0/0/1)	5 (0/1/1)
	TAS	Dn	4	0	4 (0/0/0)	4 (0/1/0)
**	TAS	Mem	3	0	12 (1/0/1)	12 (1/1/1)
	TST	Dn	0	0	2 (0/0/0)	2 (0/1/0)
*	TST	Mem	0	0	2 (0/0/0)	2 (0/1/0)

＊　実効アドレスのフェッチ時間を加算します。
＊＊　実効アドレスの計算時間を加算します。

11.6.12　シフト/ローテイト命令

シフト/ローテイト命令の実行時間の表は、プロセッサが指定されたアドレッシング・モードで、指定された操作を実行するのに必要なクロック周期数を示しています。脚注に、適切な実効アドレス時間を加算する必要がある場合を示します。特に記述がないかぎり、シフトされるビット数は実行時間に影響を与えません。命令-キャッシュ・ケースおよびノー・キャッシュ・ケースでは、合計クロック・サイクル数はかっこの外側に記入されています。リード、プリフェッチ、およびライト・サイクル数は、かっこ内に(r/p/w)の形式で記載されています。これらは合計クロック・サイクル数に含まれています。

すべての実行時間データは、2クロックのリードおよびライトを想定しています。

	命令		ヘッド	テール	Iキャッシュ・ケース	ノー・キャッシュ・ケース
	LSd	#⟨data⟩,Dy	4	0	4 (0/0/0)	4 (0/1/0)
%	LSd	Dx,Dy	6	0	6 (0/0/0)	6 (0/1/0)
+	LSd	Dx,Dy	8	0	8 (0/0/0)	8 (0/1/0)
*	LSd	Mem by 1	0	0	4 (0/0/1)	4 (0/1/1)
	ASL	#⟨data⟩,Dy	2	0	6 (0/0/0)	6 (0/1/0)
	ASL	Dx,Dy	4	0	8 (0/0/0)	8 (0/1/0)
*	ASL	Mem by 1	0	0	6 (0/0/1)	6 (0/1/1)
	ASR	#⟨data⟩,Dy	4	0	4 (0/0/0)	4 (0/1/0)
%	ASR	Dx,Dy	6	0	6 (0/0/0)	6 (0/1/0)
+	ASR	Dx,Dy	10	0	10 (0/0/0)	10 (0/1/0)
*	ASR	Mem by 1	0	0	4 (0/0/1)	4 (0/1/1)
	ROd	#⟨data⟩,Dy	4	0	6 (0/0/0)	6 (0/1/0)
	ROd	Dx,Dy	6	0	8 (0/0/0)	8 (0/1/0)
*	ROd	Mem by 1	0	0	6 (0/0/1)	6 (0/1/1)
	ROXd	Dn	10	0	12 (0/0/0)	12 (0/1/0)
*	ROXd	Mem by 1	0	0	4 (0/0/0)	4 (0/1/0)

d　シフト/ローテイトの方向；LまたはR
*　実効アドレスのフェッチ時間を加算します。
%　シフト数がデータ・サイズと等しいかそれ以下であることを示します。
+　シフト数がデータ・サイズより大きいことを示します。

11.6.13　ビット操作命令

ビット操作命令の実行時間の表は、プロセッサが指定されたアドレッシング・モードで、指定されたビット操作を実行するのに必要なクロック周期数を示しています。脚注に、適切な実効アドレス時間を加算する必要がある場合を示します。命令-キャッシュ・ケースおよびノー・キャッシュ・ケースでは、合計クロック・サイクル数はかっこの外側に記入されています。リード、プリフェッチ、およびライト・サイクル数は、かっこ内に(r/p/w)の形式で記載されています。これらは合計クロック・サイクル数に含まれています。

すべての実行時間データは、2クロックのリードおよびライトを想定しています。

命　令			ヘッド	テール	Iキャッシュ・ケース	ノー・キャッシュ・ケース
	BTST	#(data),Dn	4	0	4 (0/0/0)	4 (0/1/0)
	BTST	Dn,Dn	4	0	4 (0/0/0)	4 (0/1/0)
#	BTST	#(data),Mem	0	0	4 (0/0/0)	4 (0/1/0)
*	BTST	Dn,Mem	0	0	4 (0/0/0)	4 (0/1/0)
	BCHG	#(data),Dn	6	0	6 (0/0/0)	6 (0/1/0)
	BCHG	Dn,Dn	6	0	6 (0/0/0)	6 (0/1/0)
#	BCHG	#(data),Mem	0	0	6 (0/0/1)	6 (0/1/1)
*	BCHG	Dn,Mem	0	0	6 (0/0/1)	6 (0/1/1)
	BCLR	#(data),Dn	6	0	6 (0/0/0)	6 (0/1/0)
	BCLR	Dn,Dn	6	0	6 (0/0/0)	6 (0/1/0)
#	BCLR	#(data),Mem	0	0	6 (0/0/1)	6 (0/1/1)
*	BCLR	Dn,Mem	0	0	6 (0/0/1)	6 (0/1/1)
	BSET	#(data),Dn	6	0	6 (0/0/0)	6 (0/1/0)
	BSET	Dn,Dn	6	0	6 (0/0/0)	6 (0/1/0)
#	BSET	#(data),Mem	0	0	6 (0/0/1)	6 (0/1/1)
*	BSET	Dn,Mem	0	0	6 (0/0/1)	6 (0/1/1)

*　実効アドレスのフェッチ時間を加算します。
#　イミディエイト実効アドレスのフェッチ時間を加算します。

11. 6. 14　ビット・フィールド操作命令

ビット・フィールド操作命令の実行時間の表は、プロセッサが指定されたアドレッシング・モードを使用して、指定されたビット・フィールド操作を実行するのに必要なクロック周期数を示しています。脚注に、適切な実効アドレス時間を加算する必要がある場合を示します。命令-キャッシュ・ケースおよびノー・キャッシュ・ケースでは、合計クロック・サイクル数はかっこの外側に記入されています。リード、プリフェッチ、およびライト・サイクル数は、かっこ内に(r/p/w)の形式で記載されています。これらは合計クロック・サイクル数に含まれています。

すべての実行時間データは、2クロックのリードおよびライトを想定しています。

命令			ヘッド	テール	Iキャッシュ・ケース	ノー・キャッシュ・ケース
	BFTST	Dn	8	0	8 (0/0/0)	8 (0/1/0)
*	BFTST	Mem (<5 Bytes)	6	0	10 (1/0/0)	10 (1/1/0)
*	BFTST	Mem (5 Bytes)	6	0	14 (2/0/0)	14 (2/1/0)
	BFCHG	Dn	14	0	14 (0/0/0)	14 (0/1/0)
*	BFCHG	Mem (<5 Bytes)	6	0	14 (1/0/1)	14 (1/1/1)
*	BFCHG	Mem (5 Bytes)	6	0	22 (2/0/2)	22 (2/1/2)
	BFCLR	Dn	14	0	14 (0/0/0)	14 (0/1/0)
*	BFCLR	Mem (<5 Bytes)	6	0	14 (1/0/1)	14 (1/1/1)
*	BFCLR	Mem (5 Bytes)	6	0	22 (2/0/2)	22 (2/1/2)
	BFSET	Dn	14	0	14 (0/0/0)	14 (0/1/0)
*	BFSET	Mem (<5 Bytes)	6	0	14 (1/0/1)	14 (1/1/1)
*	BFSET	Mem (5 Bytes)	6	0	22 (2/0/2)	22 (2/1/2)
	BFEXTS	Dn	10	0	10 (0/0/0)	10 (0/1/0)
*	BFEXTS	Mem (<5 Bytes)	6	0	12 (1/0/0)	12 (1/1/0)
*	BFEXTS	Mem (5 Bytes)	6	0	18 (2/0/0)	18 (2/1/0)
	BFEXTU	Dn	10	0	10 (0/0/0)	10 (0/1/0)
*	BFEXTU	Mem (<5 Bytes)	6	0	12 (1/0/0)	12 (1/1/0)
*	BFEXTU	Mem (5 Bytes)	6	0	18 (2/0/0)	18 (2/1/0)
	BFINS	Dn	12	0	12 (0/0/0)	12 (0/1/0)
*	BFINS	Mem (<5 Bytes)	6	0	12 (1/0/1)	12 (1/1/1)
*	BFINS	Mem (5 Bytes)	6	0	18 (2/0/2)	18 (2/1/2)
	BFFFO	Dn	20	0	20 (0/0/0)	20 (0/1/0)
*	BFFFO	Mem (<5 Bytes)	6	0	22 (1/0/0)	22 (1/1/0)
*	BFFFO	Mem (5 Bytes)	6	0	28 (2/0/0)	28 (2/1/0)

＊　イミディエイト実効アドレスの計算時間を加算します。
注：32ビットのビット・フィールドが5バイトにまたがる場合はアクセスに2オペランド・サイクルが必要です。4バイトの場合は1オペランド・サイクルだけでアクセスできます。

11. 6. 15　条件分岐命令

条件分岐命令の実行時間の表は、プロセッサが指定された分岐サイズで、指定された分岐を実行するのに必要なクロック周期数を示しています。この表には全体の実行時間を示しています。これらの命令について、有効な合計実行時間を計算するのにほかの表は必要ありません。命令-キャッシュ・ケースおよびノー・キャッシュ・ケースでは、合計クロック・サイクル数はかっこの外側に記入されています。リード、プリフェッチ、およびライト・サイクル数は、かっこ内に(r/p/w)の形式で記載されています。これらは合計クロック・サイクル数に含まれています。

すべての実行時間データは、2クロックのリードおよびライトを想定しています。

命令		ヘッド	テール	Iキャッシュ・ケース	ノー・キャッシュ・ケース
Bcc	(Taken)	6	0	6 (0/0/0)	8 (0/2/0)
Bcc.B	(Not Taken)	4	0	4 (0/0/0)	4 (0/1/0)
Bcc.W	(Not Taken)	6	0	6 (0/0/0)	6 (0/1/0)
Bcc.L	(Not Taken)	6	0	6 (0/0/0)	8 (0/2/0)
DBcc	(cc = False, Count Not Expired)	6	0	6 (0/0/0)	8 (0/2/0)
DBcc	(cc = False, Count Expired)	10	0	10 (0/0/0)	13 (0/3/0)
DBcc	(cc = True)	6	0	6 (0/0/0)	8 (0/1/0)

11.6.16 制御命令

制御命令の実行時間の表は、プロセッサが指定された操作を実行するのに必要なクロック周期数を示しています。脚注に、適切な実効アドレス時間を加算する必要がある場合を示します。命令-キャッシュ・ケースおよびノー・キャッシュ・ケースでは、合計クロック・サイクル数はかっこの外側に記入されています。リード、プリフェッチ、およびライト・サイクル数は、かっこ内に(r/p/w)の形式で記載されています。これらは合計クロック・サイクル数に含まれています。

すべての実行時間データは、2クロックのリードおよびライトを想定しています。

	命令			ヘッド	テール	Iキャッシュ・ケース	ノー・キャッシュ・ケース
	ANDI to SR			4	0	12 (0/0/0)	14 (0/2/0)
	EORI to SR			4	0	12 (0/0/0)	14 (0/2/0)
	ORI to SR			4	0	12 (0/0/0)	14 (0/2/0)
	ANDI to CCR			4	0	12 (0/0/0)	14 (0/2/0)
	EORI to CCR			4	0	12 (0/0/0)	14 (0/2/0)
	ORI to CCR			4	0	12 (0/0/0)	14 (0/2/0)
	BSR			2	0	6 (0/0/1)	9 (0/2/1)
##	CAS		(Successful Compare)	1	0	13 (1/0/1)	13 (1/1/1)
##	CAS		(Unsuccessful Compare)	1	0	11 (1/0/0)	11 (1/1/0)
+	CAS2		(Successful Compare)	2	0	24 (2/0/2)	26 (2/2/2)
+	CAS2		(Unsuccessful Compare)	2	0	24 (2/0/0)	24 (2/2/0)
	CHK	Dn,Dn	(No Exception)	8	0	8 (0/0/0)	8 (0/1/0)
+	CHK	Dn,Dn	(Exception Taken)	4	0	28 (1/0/4)	30 (1/3/4)
*	CHK	EA,Dn	(No Exception)	0	0	8 (0/0/0)	8 (0/1/0)
* +	CHK	EA,Dn	(Exception Taken)	0	0	28 (1/0/4)	30 (1/3/4)
# +	CHK2	Mem,Rn	(No Exception)	2	0	18 (1/0/0)	18 (1/1/0)
# +	CHK2	Mem,Rn	(Exception Taken)	2	0	40 (2/0/4)	42 (2/3/4)
%	JMP			4	0	4 (0/0/0)	6 (0/2/0)
%	JSR			0	0	4 (0/0/1)	7 (0/2/1)
**	LEA			2	0	2 (0/0/0)	2 (0/1/0)
	LINK.W			0	0	4 (0/0/1)	5 (0/1/1)
	LINK.L			2	0	6 (0/0/1)	7 (0/2/1)
	NOP			0	0	2 (0/0/0)	2 (0/1/0)
**	PEA			0	2	4 (0/0/1)	4 (0/1/1)
	RTD			2	0	10 (1/0/0)	12 (1/2/0)
	RTR			1	0	12 (2/0/0)	14 (2/2/0)
	RTS			1	0	9 (1/0/0)	11 (1/2/0)
	UNLK			0	0	5 (1/0/0)	5 (1/1/0)

＋ 最大時間を示します。
＊ 実効アドレスのフェッチ時間を加算します。
＊＊ 実効アドレスの計算時間を加算します。
＃ イミディエイト・アドレスのフェッチ時間を加算します。
＃＃ イミディエイト・アドレスの計算時間を加算します。
% 実効アドレスへのジャンプ時間を加算します。

11. 6. 17　例外関連命令および操作

例外関連命令および操作の実行時間の表は、プロセッサが指定された例外関連の動作を実行するのに必要なクロック周期数を示しています。これらの操作に対して、有効な合計実行時間を計算するのにほかの表は必要ありません。命令-キャッシュ・ケースおよびノー・キャッシュ・ケースでは、合計クロック・サイクル数はかっこの外側に記入されています。リード、プリフェッチ、およびライト・サイクル数は、かっこ内に(r/p/w)の形式で記載されています。これらは合計クロック・サイクル数に含まれています。

すべての実行時間データは、2クロックのリードおよびライトを想定しています。

命　令	ヘッド	テール	Iキャッシュ・ケース	ノー・キャッシュ・ケース
BKPT	1	0	9 (1/0/0)	9 (1/0/0)
Interrupt (I-Stack)	0	0	23 (2/0/4)	24 (2/2/4)
Interrupt (M-Stack)	0	0	33 (2/0/8)	34 (2/2/8)
RESET Instruction	0	0	518 (0/0/0)	518 (0/1/0)
STOP	0	0	8 (0/0/0)	8 (0/2/0)
TRACE	0	0	22 (1/0/5)	24 (1/2/5)
TRAP #n	0	0	18 (1/0/4)	20 (1/2/4)
Illegal Instruction	0	0	18 (1/0/4)	20 (1/2/4)
A-Line Trap	0	0	18 (1/0/4)	20 (1/2/4)
F-Line Trap	0	0	18 (1/0/4)	20 (1/2/4)
Privilege Violation	0	0	18 (1/0/4)	20 (1/2/4)
TRAPcc (Trap)	2	0	22 (1/0/5)	24 (1/2/5)
TRAPcc (No Trap)	4	0	4 (0/0/0)	4 (0/1/0)
TRAPcc.W (Trap)	5	0	24 (1/0/5)	26 (1/3/5)
TRAPcc.W (No Trap)	6	0	6 (0/0/0)	6 (0/1/0)
TRAPcc.L (Trap)	6	0	26 (1/0/5)	28 (1/3/5)
TRAPcc.L (No Trap)	8	0	8 (0/0/0)	8 (0/2/0)
TRAPV (Trap)	2	0	22 (1/0/5)	24 (1/2/5)
TRAPV (No Trap)	4	0	4 (0/0/0)	4 (0/1/0)

11. 6. 18　セーブおよびリストア操作

セーブおよびリストア操作の実行時間の表は、プロセッサが指定された状態のセーブ、あるいは例外からの復帰を実行するのに必要な時間を示しています。この表には、全体の実行時間およびスタックの長さが記載されています。これらの操作に対して、有効な合計実行時間を計算するのにほかの表は必要ありません。命令-キャッシュ・ケースおよびノー・キャッシュ・ケースでは、合計クロック・サイクル数はかっこの外側に記入されています。リード、プリフェッチ、およびライト・サイクル数は、かっこ内に(r/p/w)の形式で記載されています。これらは合計クロック・サイクル数に含まれています。

すべての実行時間データは、2クロックのリードおよびライトを想定しています。

操　作	ヘッド	テール	Iキャッシュ・ケース	ノー・キャッシュ・ケース
Bus Cycle Fault (Short)	0	0	36 (1/0/10)	38 (1/2/10)
Bus Cycle Fault (Long)	0	0	62 (1/0/24)	64 (1/2/24)
RTE (Normal-4 Word)	1	0	18 (4/0/0)	20 (4/2/0)
RTE (Six-Word)	1	0	18 (4/0/0)	20 (4/2/0)
RTE (Throwaway)	1	0	12 (4/0/0)	12 (4/0/0)
RTE (Coprocessor)	1	0	26 (7/0/0)	26 (7/2/0)
RTE (Short Fault)	1	0	36 (10/0/0)	26 (10/2/0)
RTE (Long Fault)	1	0	76 (25/0/0)	76 (25/2/0)

11.7　アドレス変換ツリーのサーチ実行時間

アドレス変換ツリーのサーチに必要な時間は、ツリー構造の構成およびツリー内のディスクリプタ、ディスクリプタの使用済み(U)および修正(M)ビット、バス・サイクル時間、およびその他の要素によって決まります。関係する多くの変数は、サーチ時間はプログラムによって計算するのが最良であることを示唆しています。MC68030が特定の構成に対するテーブル・サーチを実行するのに必要な時間を決めるには、次の対話型プログラムを使用できます。このプログラムは、UNIX(tm) System Ⅴまたは BSD 4.2のいずれかで、sh(1)とともに使用するのに適したシェル・スクリプトです。このプログラムを使用するには、スクリプトを走らせて、システム構成と現在の状態に関する質問に答えられます。質問行の最後にある大かっこの中の数字は、キャリッジ・リターンを入力したときにプログラムが使用するデフォルト値です。

このシェル・スクリプトは、MC68030とメモリ間のデータ・バスを32ビット幅と仮定しています。これより少ないビット幅でのサーチ時間を計算するには、バス・サイクル時間のプロンプトに対し、適当なバス・サイクル時間の倍数を入力します。16ビット・データ・バスの場合、2バス・サイクルに要する時間を使用してください。また、8ビット・データ・バスの場合は、4バス・サイクルに要する時間を使用します。

このプログラムが提供する時間には、変換ツリー・サーチに要するすべての時間が含まれています。MC68030には各種のマスク・バージョンがあり、実際の時間はプログラムで計算したものとは多少異なる場合があります。

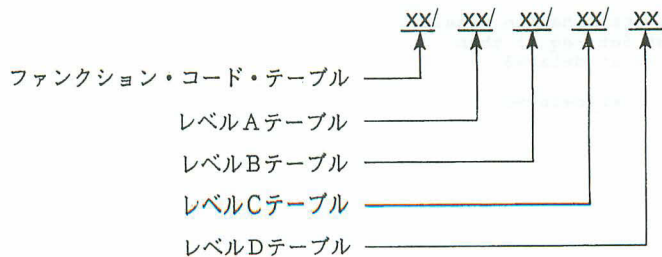
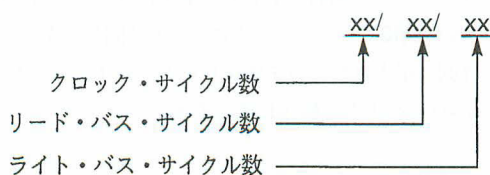
```
##########################################################################################
#
#   This Shell script is suitable for use with sh(1) on either System V or
#   BSD 4.2.  When run, it will prompt for several parameters, print a
#   configuration message, and then print the number of clocks and bus
#   cycles required for the table search.  Questions may be answered with
#   a carriage return, and the default in square brackets will be selected.
#
#   The following things should be noted by the user:
#
#   1. This script gives an approximation for the time taken for a table
#      search and associated overhead for a miss in the ATC.  The exact time
#      will vary with the instruction sequence being executed at the time of the
#      miss, and may vary plus or minus 2 clocks (see pre-walk overhead, below).
#
#   2. It will give accurate times for normal table walks (due to misses
#      in the ATC) and for PLOAD table walks but not for PTEST table walks.
#      Table walks due to the PTEST instruction will be somewhat longer.
#
#   3. It does little error checking.  It is possible to describe
#      inconsistent and impossible configurations in the script.
#
#
echo -n "Enter bus cycle time (in clocks) [2]: "
read bus
if test ! "$bus"; then
        bus=2
fi

echo -n "Enter 1 if there is a function code lookup, 0 otherwise [0]: "
read fcl
if test ! "$fcl"; then
        fcl=0
fi

echo -n "Enter number of long descriptors (page and pointer), including FCL ones [1]: "
read long
if test ! "$long"; then
        long=1
fi
```

```
echo -n "Enter number of short descriptors (page or pointer), including FCL ones [1]: "
read short
if test ! "$short"; then
        short=1
fi

echo -n "Enter 1 if there is a long indirect descriptor, 0 otherwise [0]: "
read l_ind
if test ! "$l_ind"; then
        l_ind=0
fi

echo -n "Enter 1 if there is a short indirect descriptor, 0 otherwise [0]: "
read s_ind
if test ! "$s_ind"; then
        s_ind=0
fi

echo -n "Enter number of cleared ubits encountered in pointer descriptors [0]: "
read pointer_ubits
if test ! "$pointer_ubits"; then
        pointer_ubits=0
fi

echo -n "Enter 1 if the page descriptor ubit and/or mbit is clear, 0 otherwise [0]: "

read page_m_ubit
if test ! "$page_m_ubit"; then
        page_m_ubit=0
fi

echo -n "Enter 1 if the page descriptor is encountered unexpectedly, 0 otherwise [0]: "
read et
if test ! "$et"; then
        et=0
fi

echo -n "Enter 1 if the page descriptor is long (and no rp et) [0]: "
read long_page
if test ! "$long_page"; then
        long_page=0
fi

##########################################################################################
#
# Print Configuration message.
#

levels=`expr $short + $long + $l_ind + $s_ind`

if test $fcl -eq 1; then
    tmp1=" (one for FCL)"
else
    tmp1=""
fi

out1="Configuration:  $levels levels $tmp1  - "


if test $long -ne 0 ;  then
out1="$out1 $long long descriptor(s) "
fi

if test $short -ne 0 ;  then
out1="$out1 $short short descriptor(s)"
fi

if test $l_ind -eq 1 ; then
     out1="$out1 long indirection"
elif test $s_ind -eq 1 ; then
    out1="$out1 short indirection"
fi
```

```
if test $pointer_ubits -ne 0 ;  then
    out2="$out2 $pointer_ubits  pointer ubits clear, "
fi

if test $page_m_ubit -eq 1 ; then
    out2="$out2 page ubit and/or mbit  clear, "
fi

if test $et -eq 1 ; then
    out2="$out2 early termination, "
fi

if test $long_page -eq 1 ; then
    out2="$out2 page is long;"
else
    out2="$out2 page is short;"
fi

out3="$bus clock bus cycle time."

echo
echo $out1
echo "           " $out2
echo "           " $out3
#######################################################################################
#
# Calculate result.
#
# Variables:
#
#   cough  --- the number of clocks from the start of the bus cycle that will miss to
#              the first clock of the first micro-instruction.
#
#   startup -- microcode startup overhead common to all flows
#
#   termination -- microcode termination overhead common to all flows
#
#   bus_max_4 bus_max_3  the maximum value of the bus cycle time (in clocks) and
#                        4 or 3, respectively.
#
#

bus_reads=0
bus_writes=0
ind_clocks=0

# time from BEGINNING of bus cycle which misses to first box
# this is 6 to 9 clocks depending on i- and d-state at miss-- use 7 as average
cough=7

# 4 boxes of startup, when no FCL.
startup=8

# 4 boxes of termination.
termination=8

# Bus accesses begin sooner if FCL - no limit check.
if test $fcl -eq 1 ; then
    startup=`expr $startup - 2`
fi

# calculate max((bus-4),0) for overlap
bus_max_4=`expr $bus - 4`

if test $bus_max_4 -lt 0; then
         bus_max_4=0
fi
```

```
# calculate max((bus-3),0) for overlap
bus_max_3='expr $bus - 3'

if test $bus_max_3 -lt 0; then
        bus_max_3=0
fi

overhead='expr $cough + $startup + $termination'

# number of clock due to long descriptors
l_clocks='expr $long \* \( 6 + $bus + $bus_max_4 \)'

#long page is one box less than long pointer
if test $long_page -eq 1; then
        l_clocks='expr $l_clocks - 2'
fi

bus_reads='expr $bus_reads + \( $long \* 2 \)'

# number of clock due to short descriptors
s_clocks='expr $short \* \( 3 + $bus \)'
bus_reads='expr $bus_reads + $short'

# total clocks due to descriptor fetches
t_clocks='expr $l_clocks + $s_clocks'

if test $t_clocks -eq 0 ; then
    if test $et -ne 1 ; then
        echo Error: 0 bus accesses must imply unexpected page encountered.
    fi
    et=0
fi

# now caculate clocks due to setting u bits in pointer descriptor

u_clocks='expr $pointer_ubits \* \( 4 + $bus_max_3 \)'
bus_writes='expr $bus_writes + $pointer_ubits + $page_m_ubit'

# clocks due to setting u/m bits in page descriptor
page_clocks='expr $page_m_ubit \* \( 2 + $bus_max_3 \)'
bus_writes='expr $bus_writes + $page_m_ubit'

# clocks due to indirect level (long)
if test $l_ind -ne 0; then
        ind_clocks='expr 2 + \( $bus \* 2 \)'
        bus_reads='expr $bus_reads + 2'
fi

# clocks due to indirect level (short)
if test $s_ind -ne 0; then
        ind_clocks='expr 3 + $bus'
        bus_reads='expr $bus_reads + 1'
fi

# early termination penalty
if test $et -eq 1; then
        et_delay=3
else
        et_delay=0
fi
```

```
###############################################################################
#
# Perform the calculation.
#

clocks=`expr $overhead  \
       + $l_clocks       \
       + $s_clocks       \
       + $u_clocks       \
       + $page_clocks    \
       + $ind_clocks     \
       + $et_delay`


out="    Clocks required (from beginning of missed bus cycle): $clocks"
echo
echo $out

write_accesses=`expr $pointer_ubits + $page_m_ubit`

out="    Bus Reads:                                               $bus_reads"
echo $out

print_total=0
if test $write_accesses -ne 0 ; then
    out="    Bus Writes:                                              $write_accesses"
    echo $out
    print_total=1
fi


bus_accesses=`expr $bus_reads + $write_accesses`

if test $print_total -eq 1 ; then
    out="    Total Bus Cycles:                                        $bus_accesses"
    echo $out
fi
```

以下の表では、シェル・スクリプトを使用して得られるサンプル時間をいくつか記載しています。表の各行は変換テーブルの構成を示します。各行にある識別子には、5つの位置があります。各位置はそのレベルにテーブルがないことを意味する“x”、そのレベルのテーブルがショート・フォーマットのディスクリプタで構成されていることを意味する“S”、あるいはそのレベルのテーブルがロング・フォーマットのディスクリプタで構成されていることを意味する“L”のいずれかになっています。エントリのフォーマットは次のとおりです。

テーブルの各エントリは、テーブル・サーチに必要なクロック・サイクル数、バス・リード数、およびバス・ライト数を示す3つの数値で構成されています。Uビットをセットするための RMC サイクルは、1リードおよび1ライトとしてカウントされます。記述形式は次のとおりです。

この表は以下の仮定事項に基づいて計算されています。

1. バス・サイクル時間は2クロック・サイクル
2. 間接ディスクリプタはない
3. 予期しないページ・ディスクリプタに出会うことはない(アーリ・ターミネーションはない)
4. メモリ・ポートは32ビット幅

テーブルのフォーマット	UおよびMの全ビットをセット	ページのUおよびMビットだけをセット	UおよびMビットはセットしない
LLxxx	41/4/2	37/4/1	35/4/0
LLLxx	53/6/3	45/6/1	43/6/0
LLLLx	65/8/4	53/8/1	51/8/0
LLLLL	77/10/5	61/10/1	59/10/0
SSxxx	37/2/2	33/2/1	31/2/0
SSSxx	46/3/3	38/3/1	36/3/0
SSSSx	55/4/4	43/4/1	41/4/0
SSSSS	64/5/5	48/5/1	46/5/0
xSSxx	39/2/2	35/2/1	33/2/0
xSLxx	40/3/2	36/3/1	34/3/0
xLSxx	42/3/2	38/3/1	36/3/0
xLLxx	43/4/2	39/4/1	37/4/0
xSSSx	48/3/3	40/3/1	38/3/0
xSSLx	49/4/3	41/4/1	39/4/0
xSLSx	51/4/3	43/4/1	41/4/0
xSLLx	52/5/3	44/5/1	44/5/0
xLSSx	51/4/3	43/4/1	41/4/0
xLSLx	52/5/3	44/5/1	42/5/0
xLLSx	54/5/3	46/5/1	44/5/0
xLLLx	55/6/3	47/6/1	45/6/0

11. 7. 1　MMUの実効アドレス計算

MMU命令に対する実効アドレス計算時間の表は、プロセッサが各種の実効アドレスを計算するのに必要なクロック周期数を示しています。フェッチ時間は、メモリ間接アドレッシング・モードの第1レベルの間接アドレッシングに対する時間だけを含んでいます。合計クロック・サイクル数はかっこの外側に記入されています。リード、プリフェッチ、およびライト・サイクル数は、かっこ内に(r/p/w)の形式で記載されています。

アドレス・モード	ヘッド	テール	Iキャッシュ・ケース	ノー・キャッシュ・ケース
(An)	4+op head	0	4 (0/0/0)	4 (0/1/0)
(d_{16},An)	4+op head	0	4 (0/0/0)	4 (0/1/0)
(xxx).W	4+op head	0	4 (0/0/0)	4 (0/1/0)
(xxx).L	6+op head	0	6 (0/0/0)	6 (0/2/0)
(d_8,An,Xn)	4+op head	0	4 (0/0/0)	4 (0/1/0)
全フォーマット拡張ワード				
(d_{16},An)	4	0	8 (0/0/0)	8 (0/2/0)
(d_{16},An,Xn)	4	0	8 (0/0/0)	8 (0/2/0)
([d_{16},An])	4	0	12 (1/0/0)	12 (1/2/0)
([d_{16},An],Xn)	4	0	12 (1/0/0)	12 (1/2/0)
([d_{16},An],d_{16})	2	0	12 (1/0/0)	12 (1/2/0)
([d_{16},An],Xn,d_{16})	4	0	12 (1/0/0)	12 (1/2/0)
([d_{16},An],d_{32})	4	0	14 (1/0/0)	14 (1/3/0)
([d_{16},An],Xn,d_{32})	4	0	14 (1/0/0)	14 (1/3/0)
(B)	8+op head	0	8 (0/0/0)	8 (0/1/0)
(d_{16},B)	6	0	10 (0/0/0)	10 (0/2/0)
(d_{32},B)	6	0	16 (0/0/0)	16 (0/2/0)
([B])	6	0	12 (1/0/0)	12 (1/1/0)
([B],I)	6	0	12 (1/0/0)	12 (1/1/0)
([B],d_{16})	6	0	12 (1/0/0)	12 (1/2/0)
([B],I,d_{16})	6	0	12 (1/0/0)	12 (1/2/0)
([B],d_{32})	6	0	14 (1/0/0)	14 (1/2/0)
([B],I,d_{32})	6	0	14 (1/0/0)	14 (1/2/0)
([d_{16},B])	6	0	14 (1/0/0)	14 (1/2/0)
([d_{16},B],I)	6	0	14 (1/0/0)	14 (1/2/0)
([d_{16},B],d_{16})	6	0	14 (1/0/0)	14 (1/2/0)
([d_{16},B],I,d_{16})	6	0	14 (1/0/0)	14 (1/2/0)
([d_{16},B],d_{32})	6	0	16 (1/0/0)	16 (1/3/0)
([d_{16},B],I,d_{32})	6	0	16 (1/0/0)	16 (1/3/0)
([d_{32},B])	6	0	20 (1/0/0)	20 (1/2/0)
([d_{32},B],I)	6	0	20 (1/0/0)	20 (1/2/0)
([d_{32},B],d_{16})	6	0	20 (1/0/0)	20 (1/3/0)
([d_{32},B],I,d_{16})	6	0	20 (1/0/0)	20 (1/3/0)
([d_{32},B],d_{32})	6	0	22 (1/0/0)	22 (1/3/0)
([d_{32},B],I,d_{32})	6	0	22 (1/0/0)	22 (1/3/0)

B＝ベース・アドレス；0、An、Xn、An＋Xn。フォームは実行時間に影響を与えません。
I＝インデックス；0、Xn
＊　実行時間において実効アドレスと操作を分離することはできません。ヘッドおよびテイルは操作のものです。
注：Xnを同時にBとIに入れることはできません。Xnのスケーリングおよびサイズは実行時間に影響を与えません。

11.7.2　MMU命令実行時間

MMU命令実行時間の表は、MMUがMMU命令を実行するのに必要なクロック周期数を示しています。合計クロック・サイクル数はかっこの外側に記入されています。リード、プリフェッチ、およびライト・サイクル数は、かっこ内に(r/p/w)の形式で記載されています。

命　令	ヘッド	テール	Iキャッシュ・ケース	ノー・キャッシュ・ケース
PMOVE (CRP、SRPから)*	0	0	4 (0/0/2)	5 (0/1/2)
PMOVE (CRP、SRPへ、有効)*	0	0	12 (2/0/0)	14 (2/2/0)
PMOVE (CRP、SRPへ、無効) 1*	0	0	28 (3/0/4)	30 (3/2/4)
PMOVE (TT0、TT1から)*	0	0	8 (0/0/1)	8 (0/1/1)
PMOVE (TT0、TT1へ)*	0	0	12 (1/0/0)	14 (1/2/0)
PMOVE (MMUSRから)*	2	0	4 (0/0/1)	5 (0/1/1)
PMOVE (MMUSRへ)*	0	0	6 (1/0/0)	6 (1/1/0)
PMOVE (TCから)*	2	0	4 (0/0/1)	5 (0/1/1)
PMOVE (TCへ、有効) 2*	0	0	38 (1/0/0)	40 (1/2/0)
PMOVE (TCへ、無効) 3*	0	0	56 (2/0/4)	58 (2/2/4)
PMOVE (TCへ) 4*	0	0	14 (1/0/0)	16 (1/2/0)
PFLUSHA	0	0	12 (0/0/0)	14 (0/2/0)
PFLUSH<fc>、#<mask>(fcはイミディエイトまたはデータ・レジスタ)	0	0	16 (0/0/0)	18 (0/2/0)
PFLUSH<fc>、#<mask>(fcはSFCまたはDFCレジスタ)	0	0	20 (0/0/0)	22 (0/2/0)
PFLUSH<fc>、#<mask>、<ea>(fcはイミディエイトまたはデータレジスタ)*	0	0	16 (0/0/0)	18 (0/2/0)
PFLUSH<fc>、#<mask>、<ea>(fcはSFCまたはDFCレジスタ)*	0	0	20 (0/0/0)	22 (0/2/0)
PLOAD[R:W]<fc>、<ea>(fcはイミディエイトまたはデータ・レジスタ)**	0	0	8 (0/0/0)	10 (0/2/0)
PLOAD[R:W]<fc>、<ea>(fcはSFCまたはDFCレジスタ)**	0	0	12 (0/0/0)	14 (0/2/0)
PTEST [R:W] <fc>、<ea>、#6* ***	0	0	88 (12/0/0)	88 (12/1/0)
PTEST [R:W] <fc>、<ea>、#0*	0	0	22 (0/0/0)	22 (0/1/0)

注：1. 無効なルート・ポインタをロードしようとした場合です。
　　2. 変換がイネーブルされています。
　　3. この値は最大で、ページ・サイズが有効であると仮定していますが、TIxフィールドは32までは加算しません。変換はイネーブルされています。
　　4. 変換は禁止されています。

* 該当する実効アドレスの計算時間を加算します。
** 該当する実効アドレスの計算時間とテーブル・サーチ時間を加算します。
*** この値は6レベル(FCルックアップ、a、b、c、dレベル、および間接ディスクリプタ。すべてロング・フォーマットディスクリプタの場合) テーブルに対する最大値です。

11.8　割込み待ち時間

リアルタイム・システムでは、プロセッサが割込みをサービスするのに必要な応答時間が全体的なシステム性能に関係する重要な要素です。M68000ファミリのプロセッサは、割込みの非同期でのアサートをサポートし、後続の命令境界でそれらの処理を開始します。平均割込み待ち時間は非常に短いものですが、最大待ち時間はリアルタイム割込みが最大割込み待ち時間内にサービスを要求できないため、重要になることがよくあります。MC68030の最大割込み待ち時間は、約200クロック・サイクル(MOVEM.L([d_{32}, An], Xn, d_{32})、D0-D7/A0-A7命令で、最後のデータ・フェッチがバス・エラーでアボートされた場合)ですが、MMUがイネーブルされているときには、実行に通常の数倍時間がかかる操作もあります。

MMUを使用しているシステムでの割込み待ち時間は、メイン・プロセッサの命令の長さ、アドレ

ス変換ツリーの構成、その命令で要求される変換ツリー・サーチ数、メイン・メモリのアクセス時間数、およびMC68030をメイン・メモリに接続するデータ・バス幅によって影響を受けます。ここで重要なのは、アドレス変換ツリー構成はソフトウェアの制御下にあり、システムの割込み待ち時間に大きく影響を与えるということです。あるシステム構成の最大割込み待ち時間は、最も長いメイン・プロセッサ命令の長さを、その命令が必要とする最大変換ツリー・サーチ数に必要な時間に加算することによって計算できます。MC68030マイクロプロセッサでは、特に興味深い命令は、すべてのコードおよびデータ・アイテムがページ境界をまたがる、ソースおよびデスティネーションの両方に対するメモリ間接アドレッシング付きメモリ間転送命令です。この命令のアセンブラ・シンタックスは次のようになります。

MOVE.L(od, [bd, An, Rm]), (od, [bd, An, Rm])

この命令によって、10回のアドレス変換ツリー・サーチが発生する可能性があります。すなわち、命令ストリームに2回、ソース間接アドレスに2回、デスティネーション間接アドレスに2回、オペランド・フェッチに2回、そしてデスティネーション・ライトに2回です。システム・ソフトウェアは、生成されるコードに制限を付加することによって、最大変換サーチ数を減らすことができます。たとえば、システムの言語トランスレータがロング・ワード境界に整列したロング・ワードしか生成しない場合は、間接アドレスおよびオペランドが、それぞれ1回の変換サーチしか行なわないことになる可能性があります。これによって、その命令に対するサーチ数が最大6に減ります。

11.9 バス調停待ち時間

MMUを使用するシステムでは、いくつかの要因によってバス調停待ち時間が影響を受けます。MC68030はリード・モディファイ・ライト操作の実行中は、物理バスを解放しません。アドレス変換サーチは、拡張リード・モディファイ・ライト操作であるため、システムが要求する最長アドレス変換サーチでは、ノー・キャッシュ・ケース待ち時間が発生します。

コプロセッサまたは他のデバイスが$\overline{\text{DSACKx}}$または$\overline{\text{STERM}}$信号のアサートを遅らせたり、行なわなかったときには、さらにバス調停待ち時間が追加されます。この場合の最大遅延は不定であり、その信号をアサートするときの遅延時間によって異なります。

第 12 章

アプリケーション情報

本章ではMC68030を使用するためのガイドラインを示します。最初に、MC68030をMC68020の設計に適応させるために必要な諸条件について考察します。次に、MC68881およびMC68882コプロセッサをMC68030と組み合わせて使用する方法を述べます。続いて、バイト選択ロジックを説明し、そのあとメモリ・インタフェースについて解説します。さらに後半の部分では、外部キャッシュの説明、$\overline{\text{STATUS}}$および$\overline{\text{REFILL}}$信号、そして電源およびグランドの考慮事項を述べます。

12. 1　MC68020システムへのMC68030の適応

おそらく、はじめてMC68030を利用しようとする際に最も簡単にとりかかる方法は、MC68020用に設計されたシステムをよく検討してみることでしょう。MC68020とMC68030の非同期バスは完全に互換性があるため、このアプローチが可能です。ここでは、MC68020をベースにした既存のシステムに、MC68030を実装可能にするアダプタの構成方法を説明します。この2つのプロセッサのソフトウェアとアーキテクチャ上の違いについても検討します。MC68030とMC68020はピン・コンパチブルではありませんので必ずアダプタが必要です。アダプタ・ボードを使用すれば、プログラマーズ・モデルおよびMC68030の命令セットをすぐに評価し、MC68030に追加された強力な機能を生かしたソフトウェアの開発が可能です。また、アダプタ・ボードを使用すれば、データ・キャッシュおよびMMUを内蔵したより高性能の32ビットMPUを装着することができるため、比較的簡単に既存のMC68020システムまたはMC68020/MC68851システムの性能を向上させることができます。ただし、このアダプタ・ボードはMC68030の同期バス・インタフェースをサポートしていませんので、この方式で使用するMC68030をMC68030専用に設計されたシステムと比較するときは、その点を考慮して性能評価を行なってください。

アダプタ・ボードは、MC68020ターゲット・システムのCPUソケットに差し込み、MC68030とコンパチブルにする方法で、ソケットを通して電源、グランド、およびクロック信号を引き込みます。必要なサポート用ハードウェアは、1KΩのプルアップ抵抗1本と、アダプタ・ボードの電源とグランドをデカップリングするためのコンデンサ2個だけです。

12. 1. 1　信号のルーチング

図12-1にMC68030の信号をMC68020のヘッダに配線するための回路図を示します。両方のプロセッサに共通な信号は、他方のプロセッサの対応する信号に直接配線されています。MC68030の信号のうち、MC68020の信号と互換性のないものはプルアップされるか、あるいは接続されていません。

プルアップ：$\overline{\text{STERM}}$　$\overline{\text{CIIN}}$　非接続：$\overline{\text{STATUS}}$　$\overline{\text{CBREQ}}$
$\overline{\text{CBACK}}$　$\overline{\text{MMUDIS}}$　$\overline{\text{REFILL}}$　$\overline{\text{CIOUT}}$

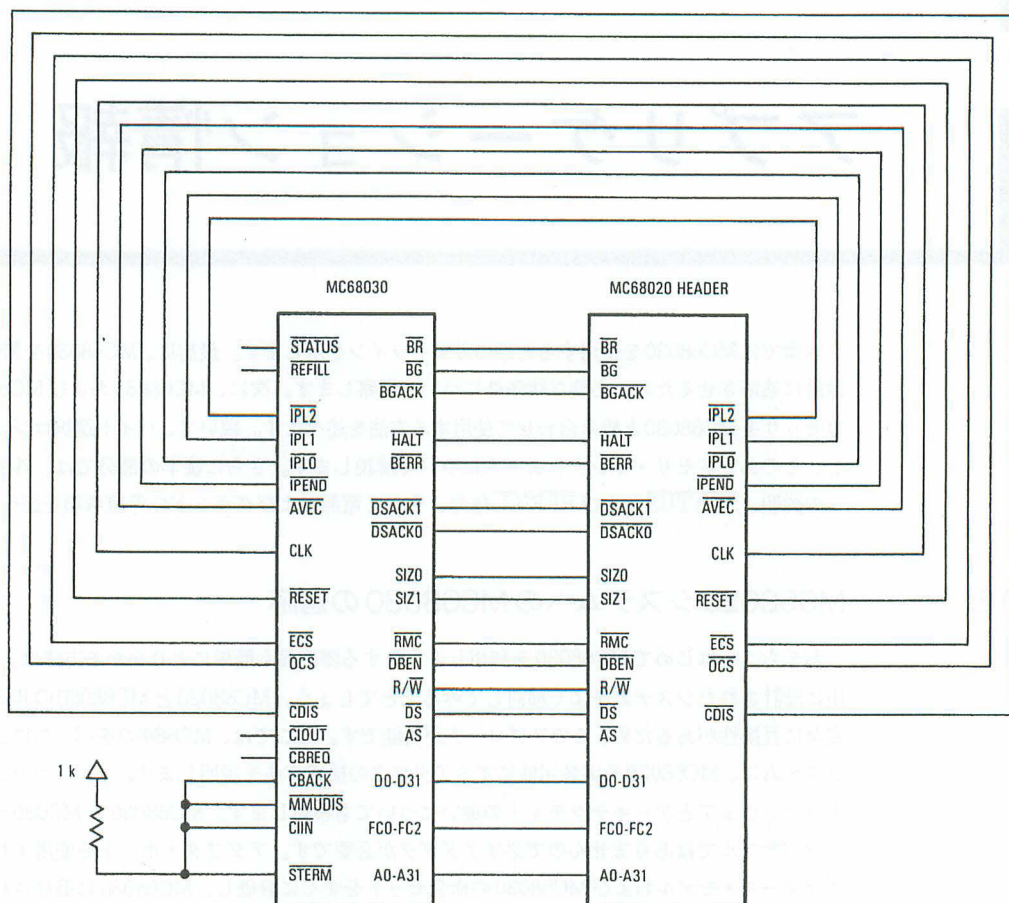



図12-1　MC68030をMC68020設計に適合させる信号ルーチング

12. 1. 2　ハードウェアの相違

MC68030のオンチップ・キャッシュをイネーブルする前に、重要なシステム機能をチェックしなければなりません。MC68030のキャッシュの構成およびインプリメンテーションによって、キャッシュ可能なリード・バス・サイクルは、SIZxピンが実際に何バイト要求していようと関係なく、データの全ポート幅($\overline{\text{DSACKx}}$のエンコーディングで示す)を転送する必要があります。MC68020にはこの条件がないため、システム・メモリ・バンクまたは周辺デバイスは、MC68030が必要とするデータを供給できる場合とできない場合があります。ターゲット・システムが、キャッシュ可能な命令またはデータ・アクセスに対して、全ポート幅の有効データを供給しない場合、ユーザはそのメモリ領域をキャッシュ不能として指定する(MMUにより)か、対応するオンチップ・キャッシュをイネーブルにしないようにしなければなりません。システムによっては、ターゲット・システムのハードウェアを変更するようになってくる場合もあります。1個のPALでバイト選択ロジックを作って

いることもよく見られます。PALは簡単に交換したり再プログラムできるため、リード・サイクル中にマルチバイト・ポートから全バイトを選択できるようにすることもできるわけです。

MC68030にある$\overline{\text{HALT}}$信号は入力専用信号となっており、MC68020の双方向性$\overline{\text{HALT}}$信号とはいくぶん異なります。しかし、外部システムに対して、ダブル・バス・フォールトのためにプロセッサが停止したという表示(たとえば、LED を点灯して)がなくなる以外、それによって問題が発生することはありません。

本来MC68020およびMC68851用に設計されたシステムで使用する場合、MC68851はそのままシステムに残しておいてもよく、また取り外して(ジャンパ配線を施したヘッダに差し替えておく)もかまいません。ただし、システムに残しておいても、このMC68851にM68000コプロセッサ・インタフェースを通してプログラマがアクセスすることはできません。すべてのMMU命令は、MC68030のオンチップMMUをアクセスします。MC68030の$\overline{\text{MMUDIS}}$信号がアサートされた場合でも同じです。MC68851を取り外した場合の利点は、物理バスに対する非同期バス・サイクルの最小時間が4クロック・サイクルから3クロック・サイクルに減少することです。

MC68851を取り外して、ジャンパ配線されたヘッダに置き換えた場合、MC68851の信号のうち、$\overline{\text{CLI}}$、$\overline{\text{RMC}}$、$\overline{\text{LBRO}}$、$\overline{\text{LBG}}$、$\overline{\text{LBGACK}}$、および$\overline{\text{LBGI}}$については、それぞれのシステムに応じて検討する必要があります。変換テーブル・サーチ中に、MC68851は$\overline{\text{CLI}}$(キャッシュ・ロード・インヒビット)信号をアサートしますが$\overline{\text{RMC}}$はアサートしません。一方、MC68030は$\overline{\text{RMC}}$はアサートしますが$\overline{\text{CIOUT}}$はアサートしません。論理バス調停や論理キャッシュのない簡単なMC68020/MC68851システムでは、MC68851のジャンパに以下の信号を結合することができます。

$\overline{\text{LAS}} \longleftrightarrow \overline{\text{PAS}}$
$\overline{\text{LBRO}} \longleftrightarrow \overline{\text{PBR}}$
$\overline{\text{LBGI}} \longleftrightarrow \overline{\text{PBG}}$
$\overline{\text{LBGACK}} \longleftrightarrow \overline{\text{PBGACK}}$
LA(8-31)$\longleftrightarrow$PA(8-31)
$\overline{\text{CLI}} \longleftrightarrow$接続なし、または$\overline{\text{LAS}}$

$\overline{\text{CLI}}$には2つの接続オプションがありますが、これはMC68851の$\overline{\text{PAS}}$がアサートされないので、一部のシステムは$\overline{\text{CLI}}$を使用してCPU空間サイクルの発生を認可しているためです。

12.1.3　ソフトウェアの相違

MC68030のキャッシュ制御レジスタ(CACR)の命令キャッシュ制御ビットは、MC68020のCACRの対応するビットと同じビット位置にあります。しかし、MC68030にはバースト・イネーブルおよびデータ・キャッシュ制御用として、さらに別の制御ビットがあります。このアダプタ・ボードは同期バス・サイクル(したがって、バースト・モード)をサポートしないため、CACRによってバースト・モードをイネーブルしても、システム動作にはまったく影響を与えません。ビット位置およびCACRビットの機能の詳細については、「第6章　オンチップ・キャッシュ・メモリ」を参照してください。

MC68020用に設計されたシステムで使用するときに、プログラマが認識しておかなければならない違いは、MC68030はMC68020のCALLMおよびRTM命令をサポートしていないことです。CALLM命令またはRTM命令を使用したコードをMC68030で実行すると、未実装命令例外が発生します。ユーザは、MMUのソフトウェア開発機能を必要とせず、ハードウェアの相違のところで述べたキャッシュの動作が理解できれば、MC68030のMMUを無視することができます。

本来MC68020/MC68851ペア用に設計されたシステムでアダプタを使用するときは、以下に述べるソフトウェアの相違も当てはまります。MC68030のMMUは、MC68851の機能のサブセットを備えています。MC68030のMMUがサポートしていない機能は次のとおりです。

第12章　アプリケーション情報

- オンチップ・ブレークポイント・レジスタ
- タスクの別名付け
- 命令：PBcc, PDBcc, PRESTORE, PSAVE, PScc, PTRAPcc, PVALID

MC68030のMMU命令には、制御-可変アドレッシング・モードしか許されていません。

MC68030のMMUに新しく追加された機能(MC68851にはないもの)は、トランスペアレント変換レジスタによる、2つの論理アドレス・ブロックのトランスペアレント変換です。これについては、「第9章　メモリ管理ユニット」を参照してください。

12.2　浮動小数点ユニット

MC68030の浮動小数点サポートは、MC68881浮動小数点コプロセッサおよびMC68882高性能浮動小数点コプロセッサによって提供されます。両方のデバイスとも2進浮動小数点演算のためのIEEE規格(754)に完全に適合しています。MC68882はMC68881とピンおよびソフトウェア・コンパチブルの機能強化デバイスで、同じクロック周波数でMC68881の1.5倍を上回る性能が得られる最適化MPUインタフェースを備えています。

両方のコプロセッサとも、メイン・プロセッサの整数データ処理能力を論理的に拡張します。高性能浮動小数点演算ユニット、およびプロセッサの整数データ・レジスタと同様に使用できる浮動小数点データ・レジスタ群を内蔵しています。MC68881/MC68882の命令セットは、M68000ファミリすべての初期メンバの自然拡張版であり、MC68030のすべてのアドレシング・モードおよびデータ・タイプをサポートします。プログラマにとっては、MC68030/コプロセッサの実行モデルは、あたかも両方のデバイスが1チップ上にインプリメントされているように感じられるはずです。MC68881およびMC68882は、完全なIEEE規格をサポートするだけでなく、フル・セットの三角関数および超越関数、オンチップ定数、および全80ビットの拡張精度実データ・フォーマットを提供します。

MC68030とMC68881またはMC68882とのインタフェースは、システムの価格/性能のニーズに容易に適応させることができます。MC68030とMC68881/MC68882は、M68000の標準非同期バス・サイクルによって通信を行ないます。すべてのデータ転送は、MC68881/MC68882からの要求に応じて、メイン・プロセッサが実行します。したがって、メモリ管理、バス・エラー、アドレス・エラーおよびバスの調整は、あたかもメイン・プロセッサがMC68881/MC68882の命令を実行したかのように実行されます。浮動小数点ユニットとプロセッサは、異なるクロック速度で動作させることができ、1つのMC68030システムには同時に7個の浮動小数点プロセッサが常駐可能です。

図12-2にMC68881/MC68882とMC68030とのコプロセッサ・インタフェース接続を示します(全32ビット・データ・バスを使用)。A0ピンおよび$\overline{\mathrm{SIZE}}$ピンの両方がV_{CC}に接続されているときは、MC68881/MC68882は32ビット・データ・バスに対応して動作するように構成されます。32ビット以下のデータ・バス幅でMC68881/MC68882を構成する方法については、MC68881/MC68882 のユーザーズ・マニュアルを参照してください。MC68030は、CPU空間へのアクセス中に取得したデータはキャッシュしないため、コプロセッサ・インタフェースにはMC68030キャッシュ・インヒビット入力($\overline{\mathrm{CIIN}}$)信号は使用されません。

チップ・セレクト($\overline{\mathrm{CS}}$)デコード回路は、特定の浮動小数点コプロセッサがアドレス指定されたときに検出を行なう非同期ロジックです。ロジックに使用するMC68030の信号には、ファンクション・コード信号(FC2～FC0)およびアドレス・ライン(A19～A13)が含まれています。これらの信号のエンコーディングに関する詳細は、「第10章　コプロセッサ・インタフェースの説明」を参照してください。これらのラインのすべてをデコードするか一部をデコードするかは、システムのコプロセッサの個数と、システムで許容されるマッピングの程度によって決まります。

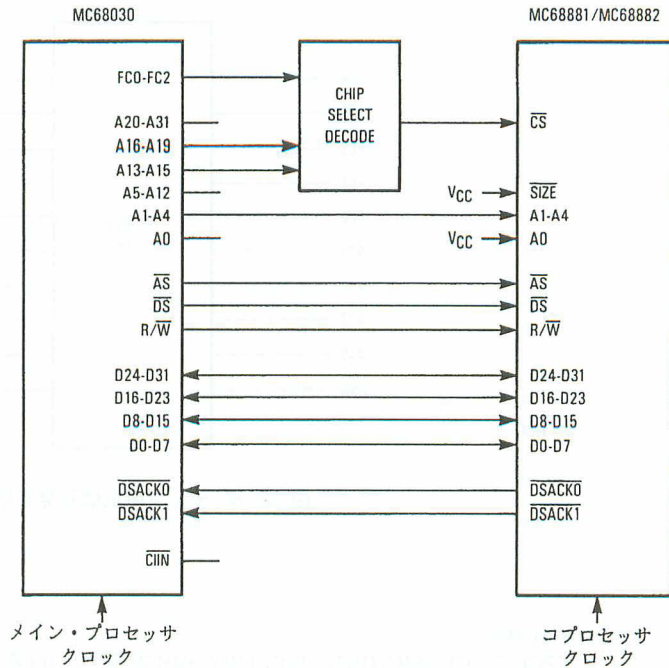



図12-2　32ビット・データ・バスのコプロセッサへの接続

システム設計者が主に注意することは、FPCPアクセスに不必要なウエイト・ステートをもたせないで、MC68030(MPU)とMC68881/MC68882(FPCP)の両方に適合するAC電気的特性を備えた$\overline{\text{CS}}$のインタフェースを設計することです。

次に示す最大規格値(CLK“L”を基準とする)がこれらの目的に適合します。

t_{CLK}“L”から$\overline{\text{AS}}$“L”≦(MPU Spec1－MPU Spec47A－FPCP Spec19)　　(1)

t_{CLK}“L”から$\overline{\text{CS}}$“L”≦(MPU Spec1－MPU Spec47A－FPCP Spec19)　　(2)

ワースト・ケースの状態では必要条件(1)は満足されませんが、MPUの$\overline{\text{AS}}$の負荷が規格値に適合し、FPCPへの$\overline{\text{AS}}$入力がバッファされていない場合は、標準状態でこの必要条件が満足されます。必要条件(2)を満足するように$\overline{\text{CS}}$発生回路を設計すれば、不必要なウエイト・ステートなしで、FPCPにアクセスできる確率が最も高くなります。図12-4の等式に従ってプログラムされた、最大伝搬遅延時間が10nsのPAL 16L8(図12-3)を使用して$\overline{\text{CS}}$を発生させることができます。この設計方法を用いる場合、25MHzシステムではt_{CLK}“L”から$\overline{\text{CS}}$“L”は10ns以下になります。ワースト・ケース状態で、t_{CLK}“L”から$\overline{\text{AS}}$“L”が必要条件(1)の値より大きくなった場合は、FPCPへのアクセスに1ウエイト・ステートが挿入されます。その他の悪影響はありません。図12-5にこのインタフェースのバス・サイクルのタイミングを示します。FPCPの仕様については、MC68881/MC68882浮動小数点コプロセッサのユーザーズ・マニュアルを参照してください。

$\overline{\text{CS}}$を発生する回路はもう1つの条件を満足していなければなりません。浮動小数点アクセスの直後に浮動小数点以外のアクセスが続く場合、$\overline{\text{CS}}$(浮動小数点アクセスに対応)をネゲートしてから、$\overline{\text{AS}}$および$\overline{\text{DS}}$(後続のアクセス)をアサートしなければなりません。上記のPAL回路はこの条件を満足しています。

たとえば、システムにコプロセッサが1個しかなければ、必ずしも図12-4のPAL等式で与えられる10本の信号(FC0～FC2およびA13～A19)を完全にデコーディングする必要はありません。FC0～FC1とA16～A17を使用するだけで十分です。

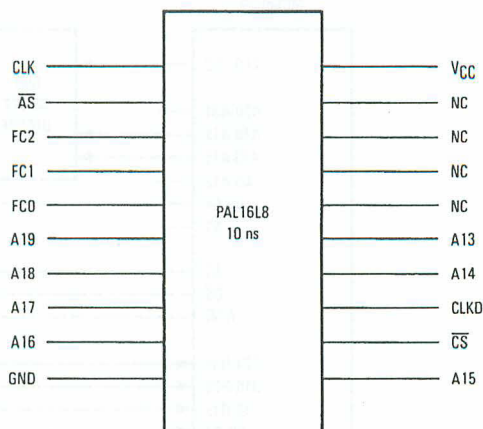



図12-3　チップ選択信号発生用PAL

```
PAL16l8
FPCP CS GENERATION CIRCUITRY FOR 25 MHz OPERATION
MOTOROLA INC., AUSTIN, TEXAS
CLK   AS    FC2    FC1   FC0   A19   A18   A17   A16   GND
A15   /CS   /CLKD  A14   A13   NC    NC    NC    NC    VCC

CS   = FC2    * FC1    * FC0            ; cpu スペース＝$7
       * /A19 * /A18   * A17   * /A16   ; コプロセッサ・アクセス＝$2
       * /A15 * /A14   * A13            ; コプロセッサid＝$1
       * /CLK                           ; MPUクロック“L”で認知する

       + FC2  * FC1    * FC0            ; cpu スペース＝$7
       * /A19 * /A18   * A17   * /A16   ; コプロセッサ・アクセス＝$2
       * /A15 * /A14   * A13            ; コプロセッサid＝$1
       * /AS                            ; アドレス・ストローブ“L”で認知する

       + FC2  * FC1    * FC0
       * /A19 * /A18   * A17   * /A16   ; コプロセッサ・アクセス＝$2
       * /A15 * /A14   * A13            ; コプロセッサid＝$1
       * /CLKD                          ; クロックD(遅延クロック)で認知する
CLKD = CLK
```

説明：CSを発生するための項目が3つあります。最初の項はCSをアサートできる最も早い時期を示します。2番目の項はFPCPのアクセスが終了するまでCSをアサートするのに使用します。3番目の項はレイトASで競合状態が発生しないようにしています。

図12-4　PAL 等式

FC1～FC0は、バス・サイクルがCPU空間($7)またはユーザ定義空間($3)のいずれかで動作していることを示し、A16～A17はCPU空間のタイプをコプロセッサ空間($2)としてデコードすることを示します。この場合、A13～A15はシステム内の複数のコプロセッサを識別するのに使用するコプロセッサ識別コード(Cp-ID)をエンコードするためのものですので、無視することができます。モトローラのアセンブラは、浮動小数点命令に対しては$1をデフォルトのCp-IDとしています。これとは別のCp-IDが必要な場合やシステムに複数のコプロセッサが存在する場合は、アセンブラのオプション機能などで制御可能です。

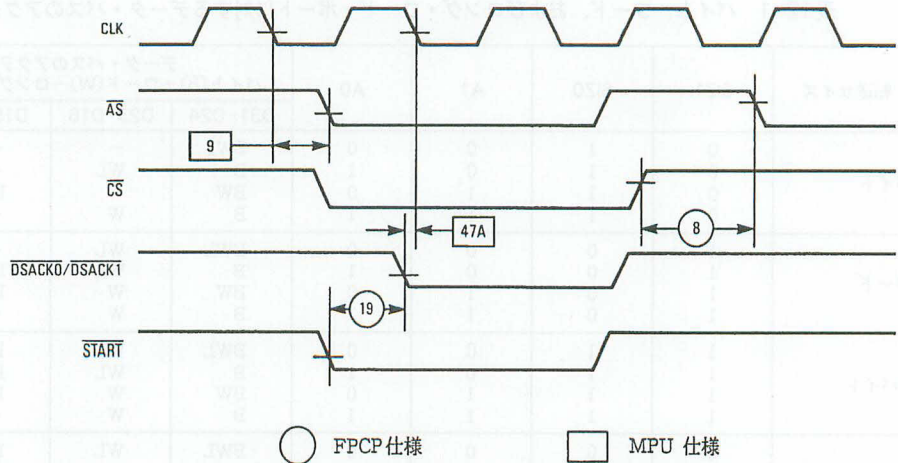



図12-5　バス・サイクルのタイミング図

12.3　MC68030のバイト選択ロジック

MC68030のアーキテクチャは、アラインメントに関係なく8ビット、16ビットあるいは32ビットのデータ・ポートへのバイト、ワード、およびロング・ワード・オペランド転送をサポートできます。この特長により、プログラマはバス幅に限定されないコードを記述することができます。周辺デバイスまたはメモリ・サブシステムは、アクセスされると、プロセッサに実際のポート・サイズを知らせます。すると、MC68030はそれに合わせ、必要な場合はマルチプル・バス・サイクルを使用して、データ転送のダイナミック・サイジングを行ないます。ハードウェアの設計者はソフトウェアの先入観にとらわれず、自由に実装方法を選択できます。以下の各項では、ダイナミック・バス・サイジング・メカニズムをイネーブルするバイト選択制御信号、異なるサイズのオペランド転送、およびミスアラインメントのオペランドを正しく動作させるための転送について説明します。

以下の信号はMC68030のオペランド転送メカニズムを制御します。

- A1、A0＝アドレス・ライン。転送するオペランドの最上位バイトを直接アドレス指定します。
- SIZ1、SIZ0＝転送サイズ。MC68030の出力です。これらはあるバス・サイクル中に転送しなければならないオペランドの残りのバイト数を示します。
- $R/\overline{W}$＝リード/ライト。MC68030の出力です。MC68030システムでバイト選択信号を発生させるには、デバイスからのデータがキャッシュ可能な場合は、$R/\overline{W}$ をロジックに組み込んでおかなければなりません。
- $\overline{DSACK1}$、$\overline{DSACK0}$＝データ転送およびサイズ・アクノリッジ信号。非同期ポートからドライブされ、ポートの実際のバス幅を示します。
- $\overline{STERM}$＝同期終了。32ビット同期ポートによってのみドライブされます。

MC68030は16ビット・ポートはデータ・ラインD16～D31、8ビット・ポートはデータ・ラインD24～D31にあるものと想定しています。これによって、それ以降のロジックがMC68030に内蔵されている内部-外部データ・バス・マルチプレクサによって正しく動作します。ダイナミック・バス・サイジング・メカニズムの詳細については、「第7章　バス操作」を参照してください。

例を見ればバイト選択信号の必要性がよくわかります。ワード構成メモリの奇数アドレスに対するロング・ワードのライト・サイクルについて検討してみましょう。この転送を完了するには、3バス・サイクルが必要です。最初のバス・サイクルでD16～D23のロング・ワードの最上位バイトを転

表12-1　バイト、ワード、およびロング・ワード・ポートに対するデータ・バスのアクティビティ

転送サイズ	SIZ1	SIZ0	A1	A0	データ・バスのアクティブ部分 バイト(B)−ワード(W)−ロング・ワード(L)ポート			
					D31-D24	D23-D16	D15-D8	D7-D0
バイト	0	1	0	0	BWL	−	−	−
	0	1	0	1	B	WL	−	−
	0	1	1	0	BW	−	L	−
	0	1	1	1	B	W	−	L
ワード	1	0	0	0	BWL	WL	−	−
	1	0	0	1	B	WL	L	−
	1	0	1	0	BW	W	L	L
	1	0	1	1	B	W	−	L
3バイト	1	1	0	0	BWL	WL	L	−
	1	1	0	1	B	WL	L	L
	1	1	1	0	BW	W	L	L
	1	1	1	1	B	W	−	L
ロング・ワード	0	0	0	0	BWL	WL	L	L
	0	0	0	1	B	WL	L	L
	0	0	1	0	BW	W	L	L
	0	0	1	1	B	W	−	L

送します。

2番目のバス・サイクルはD16～D31のワードを転送し、最後のバス・サイクルでD24～D31にあるもとのロング・ワードの最下位バイトを転送します。これらの転送に使用しないバイトに重ね書きしないようにするために、デバイスを16ビットおよび32ビット・ポート幅で使用する場合は、各バイトに対して固有のバイト・データ・ストローブを発生させなければなりません。

キャッシュ不可能なリード・サイクルおよびすべてのライト・サイクルに対しては、バス転送のために必要なデータ・バスのアクティブ・バイトは、サイズ(SIZ1/SIZ0)と下位アドレス(A1/A0)出力の関数であり、これを表12-1に示します。個々のストローブおよび選択信号は、それぞれのバス・サイクルに対して4つの信号をデコードして発生させることができます。8ビット・ポートにあるデバイスは、どの転送にも有効バイトが1つしかないため、データ・ストローブ($\overline{\mathrm{DS}}$)だけを使用することができます。

キャッシュ可能なリード・サイクルでは、アドレス指定されるデバイスは全バス幅($\overline{\mathrm{DSACKx}}$ または$\overline{\mathrm{STERM}}$で表示される)で有効なデータを供給しなければなりません。命令は常にロング・ワードに整列したアドレスとしてプリフェッチされるため、どのようなアラインメントおよびサイズでも、データ・フェッチが可能です。MC68030はデータ・バス・ポート・サイズ全体に、有効なデータがあるものと想定しているため、キャッシュ可能なデータ・リード・バス・サイクルは、バス・サイクル中にポート・サイズで通知されたデータを供給しなければなりません。この条件を満たすために、MC68030に対するバイト選択ロジックにR/$\overline{\mathrm{W}}$ 信号を組み込んでおく必要があります。

図12-6に2メモリ・バンクをもつMC68030システムのブロック図を示します。このPALは、非同期32ビット・ポートに対してメモリ・マップト・バイト選択信号、および他のメモリ・バンクやポートで使用可能な非マップト・バイト選択信号を供給します。図12-7にPALのサンプル等式を示します。

ここに示すPALの等式と回路は、あらゆるシステムに最適なものになるよう意図したものではありません。CPUのクロック周波数、メモリ・アクセス時間、およびシステムのアーキテクチャによっては、別の回路が必要な場合もあります。

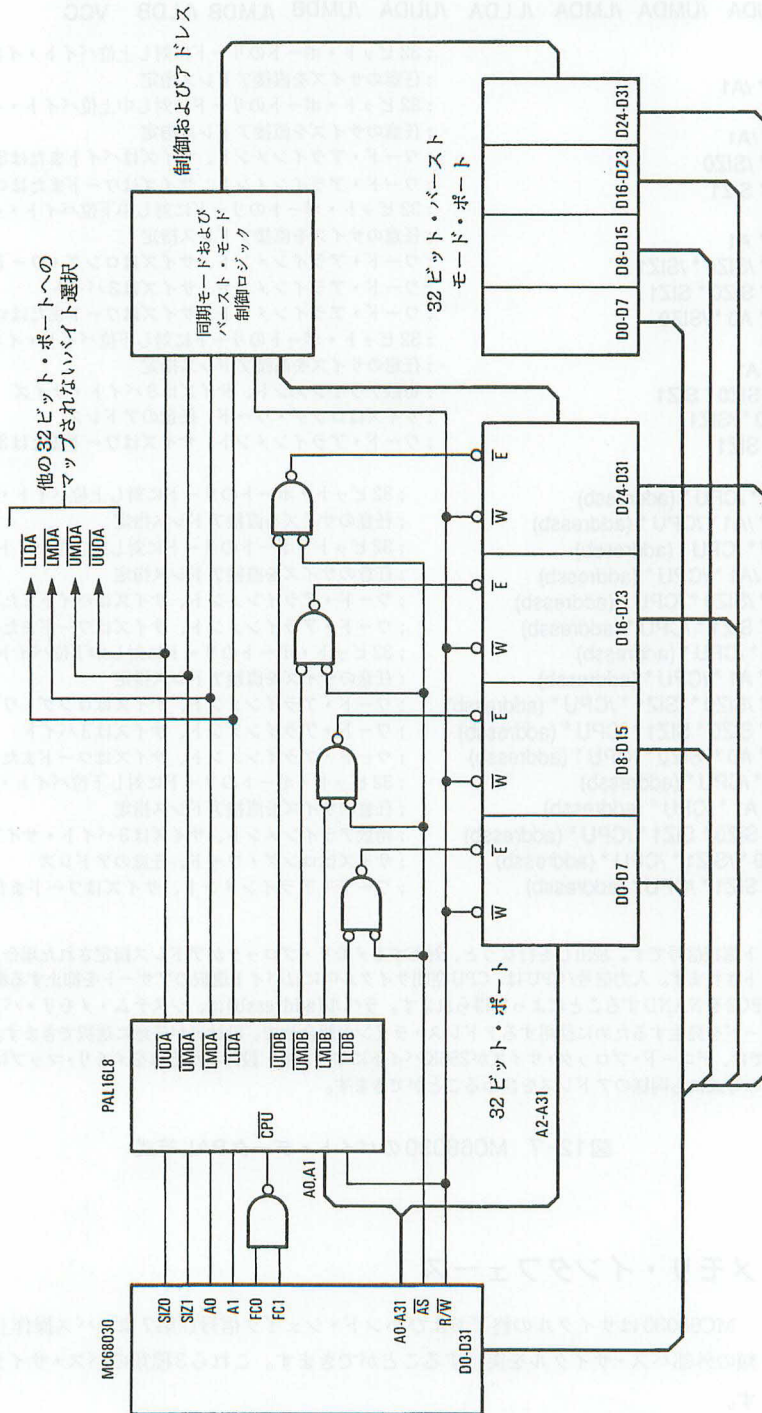



図12-6　MC68030のバイト選択PALシステムの構成例

MC68030のマップト、アンマップト32ビット・ポートに対するバイト・データ選択信号の生成

```
MOTOROLA INC., AUSTIN, TEXAS
A0    A1     SIZ0   SIZ1   RW     A18    A19    A20    A21    GND
/CPU  /UUDA  /UMDA  /LMDA  /LLDA  /UUDA  /UMDB  /LMDB  /LLDB  VCC

UUDA = RW                           ; 32ビット・ポートのリードに対し上位バイト・イネーブル
     + /A0 * /A1                    ; 任意のサイズを直接アドレス指定
UMDA = RW                           ; 32ビット・ポートのリードに対し中上位バイト・イネーブル
     + A0 * /A1                     ; 任意のサイズを直接アドレス指定
     + /A1 * /SIZ0                  ; ワード・アラインメント、サイズはバイトまたは3バイト
     + /A1 * SIZ1                   ; ワード・アラインメント、サイズはワードまたはロング・ワード
LMDA = RW                           ; 32ビット・ポートのリードに対し中下位バイト・イネーブル
     + /A0 * A1                     ; 任意のサイズを直接アドレス指定
     + /A1 * /SIZ0 * /SIZ1          ; ワード・アラインメント、サイズはロング・ワード
     + /A1 * SIZ0 * SIZ1            ; ワード・アラインメント、サイズは3バイト
     + /A1 * A0 * /SIZ0             ; ワード・アラインメント、サイズはワードまたはロング・ワード
LLDA = RW                           ; 32ビット・ポートのリードに対し下位バイト・イネーブル
     + A0 * A1                      ; 任意のサイズを直接アドレス指定
     + A0 * SIZ0 * SIZ1             ; 奇数アラインメント、サイズは3バイト・サイズ
     + /SIZ0 * /SIZ1                ; サイズはロング・ワード、任意のアドレス
     + A1 * SIZ1                    ; ワード・アラインメント、サイズはワードまたは3バイト・サイズ

UUDB = RW * /CPU * (addressb)                   ; 32ビット・ポートのリードに対し上位バイト・イネーブル
     + /A0 * /A1 * /CPU * (addressb)            ; 任意のサイズを直接アドレス指定
UMDB = RW * /CPU * (addressb)                   ; 32ビット・ポートのリードに対し中上位バイト・イネーブル
     + A0 * /A1 * /CPU * (addressb)             ; 任意のサイズを直接アドレス指定
     + /A1 * /SIZ0 * /CPU * (addressb)          ; ワード・アラインメント、サイズはバイトまたは3バイト
     + /A1 * SIZ1 * /CPU * (addressb)           ; ワード・アラインメント、サイズはワードまたはロング・ワード
LMDB = RW * /CPU * (addressb)                   ; 32ビット・ポートのリードに対し中下位バイト・イネーブル
     + /A0 * A1 * /CPU * (addressb)             ; 任意のサイズを直接アドレス指定
     + /A1 * /SIZ0 * /SIZ1 * /CPU * (addressb)  ; ワード・アラインメント、サイズはロング・ワード
     + /A1 * SIZ0 * SIZ1 * /CPU * (addressb)    ; ワード・アラインメント、サイズは3バイト
     + /A1 * A0 * /SIZ0 * /CPU * (addressb)     ; ワード・アラインメント、サイズはワードまたはロング・ワード
LLDB = RW * /CPU * (addressb)                   ; 32ビット・ポートのリードに対し下位バイト・イネーブル
     + A0 * A1 * /CPU * (addressb)              ; 任意のサイズを直接アドレス指定
     + A0 * SIZ0 * SIZ1 * /CPU * (addressb)     ; 奇数アラインメント、サイズは3バイト・サイズ
     + /SIZ0 * /SIZ1 * /CPU * (addressb)        ; サイズはロング・ワード、任意のアドレス
     + A1 * SIZ1 * /CPU * (addressb)            ; ワード・アラインメント、サイズはワードまたは3バイト・サイズ
```

説明：書込み用バイト選択信号です。読出しを行なうと、対応するメモリ・ブロックがアドレス指定された場合、すべてのバイト選択信号がアサートされます。入力信号/CPUは、CPU空間サイクル中にはバイト選択のアサートを抑止する機能をもち、FC0-FC1またはFC0-FC2をNANDすることによって得られます。ラベル(addressb)は、システム・メモリ・バンクに対する適切なアドレス・デコードを発生するために使用するアドレス・ラインの組合せで、設計者が任意に選択できます。ここに記載したアドレス・ラインでは、デコード・ブロック・サイズが256Kバイトになります。設計者がそれをメモリ・マップにした場合は、UUDA、UMDA などの等式にも同様のアドレスを含めることができます。

図12-7　MC68030のバイト・データPAL等式

12. 4　メモリ・インタフェース

MC68030はサイクルの終了およびハンド・シェイク信号(「第7章　バス操作」参照)で決まる3種類の外部バス・サイクルを実行することができます。これら3種類のバス・サイクルを以下に示します。

1. $\overline{\text{DSACKx}}$信号で終了する非同期サイクルは、最小3プロセッサ・クロック周期をもち、最大4バイトを転送します。
2. $\overline{\text{STERM}}$信号で終了する同期サイクルは、最小2プロセッサ・クロック周期をもち、最大4バイトを転送します。

3. $\overline{\text{STERM}}$および$\overline{\text{CBACK}}$信号で終了するバースト動作サイクルは、最小5プロセッサ・クロック周期をもち、最大4ロング・ワード(16バイト)を転送します。

リード操作中、M68000プロセッサは、バス・サイクルの最後のクロック・エッジ、つまりバス・サイクルが終了する0.5クロック前にデータをラッチします(バースト・モードは特別なケース)。次の立上りクロック・エッジではなく、最後のクロック・エッジでデータをラッチすることによって、次のバス・サイクルとの間でデータ・バスの競合を回避すれば、MC68030はすぐに実行ユニットの中にデータを受信することができるため、実質的な性能が向上します。ライト操作もこのデータ・バスのタイミングを使用して、データ・ホールド時間によってストローブをネゲートしないようにし、後続バス・サイクルとの間で競合が発生するのを回避します。これによって、最小限のバス・バッファおよびバス・ラッチをもつシステムを設計することができます。MC68030のオンチップ・キャッシュを使用する利点の1つは、外部メモリ構成に関係なく、キャッシュは常に“ノー・ウエイト”でアクセスされるため、外部ウエイト・ステートが性能におよぼす影響が軽減されます。MC68030(およびMC68020)はこの特長を備えているため、他の汎用プロセッサとは違っているのです。

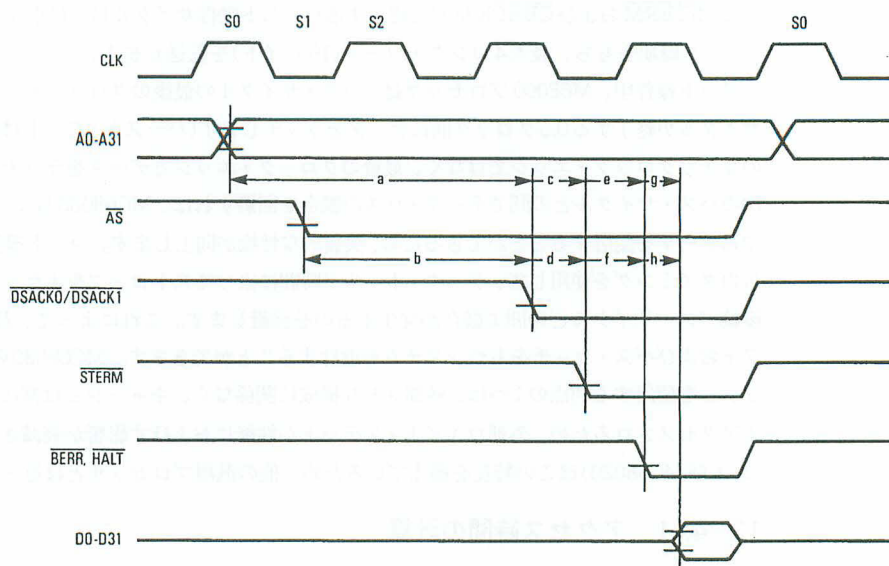
12. 4. 1　アクセス時間の計算

一般にクリティカルとなるタイミング・パスを図12-8に示します。バースト転送では、最初に転送するロング・ワードもこれらのパラメータを使用しますが、それ以降の転送では方法が異なるため、それについては「12. 4. 2　バースト・モード・サイクル」で説明します。

MC68030にインタフェースされるデバイスの種類によって、どのパスが最もクリティカルであるかが正確に決まります。スタティック・デバイスでは、通常はアドレス-データ・パスがクリティカル・パスになります。これは、スタティック・デバイスが自分のためのサイクルを開始し、後で適当な制御信号によってそのアクセスを有効にするのに何ら危険がないためです。これに対して、ダイナミック・デバイスでは、アクセスを開始する前にサイクルを有効にしなければならないことから、アドレス・ストローブ-データ有効パスが最もクリティカルになることがよくあります。性能を高めるために、データが有効になる前にバス・サイクルの終了を知らせるデバイス(たとえば、エラー検出やエラー訂正のハードウェア、または一部の外部キャッシュ)の場合、クリティカル・パスはアドレスまたはストローブから$\overline{\text{BERR}}$(または$\overline{\text{BERR}}$と$\overline{\text{HALT}}$)のアサーションまでのはずです。最後に、高速デバイスおよび外部キャッシュにとっては、アドレス有効から$\overline{\text{DSACKx}}$または$\overline{\text{STERM}}$がアサートされたパスが最もクリティカルになります。それは、アドレスが有効になってから$\overline{\text{DSACKx}}$または$\overline{\text{STERM}}$をアサートしてそのバス・サイクルを終了しなければならないまでの時間が短いためです。表12-2に、クロックのデューティ・サイクルを50％と仮定したときに、各種のメモリ・アクセス時間を計算するのに必要な等式を示します。

非同期バス・サイクルでは、$\overline{\text{DSACK1}}$と$\overline{\text{DSACK0}}$を使用して現在実行中のバス・サイクルを終了します。異なるクロック周波数で動作している周辺デバイスへのアクセスなど、真の非同期動作では、クロックに関係なく$\overline{\text{DSACK1}}$と$\overline{\text{DSACK0}}$のいずれか一方またはその両方の信号をアサートすることができます。その後データは仕様＃31で規定される時間だけ有効になっていなければなりません。クロック周波数が16.67MHzのプロセッサでは、この時間は$\overline{\text{DSACKx}}$のアサート後50nsであり、20.0MHzのプロセッサでは43nsです(これらの数値は実際のクロック周波数によって異なります)。

しかし、メモリ制御ロジックはMC68030のクロックに関連して動作するか、ワースト・ケースの伝搬遅延時間が分かっていて、それによって$\overline{\text{DSACKx}}$信号の非同期セットアップ時間を保証できるため、多くのローカル・メモリ・システムは真の非同期方式では動作しません。この擬似同期$\overline{\text{DSACKx}}$を発生するのに必要なタイミング条件は、t_{AVDL}の等式によって求めることができます。

同期サイクルでは$\overline{\text{STERM}}$信号を使用して、現在実行中のバス・サイクルを終了します。同じ長

注：この図はアクセス時間の計算だけを示します。$\overline{\text{DSACK0}}$/$\overline{\text{DSACK1}}$および$\overline{\text{STERM}}$を同じバス・サイクルで同時にアサートしてはなりません。

パラメータ	説　明	システム	等　式
a	アドレス有効→$\overline{\text{DSACKx}}$アサート	t_{AVDL}	12-1
b	アドレス・ストローブのアサート→$\overline{\text{DSACKx}}$アサート	t_{SADL}	12-2
c	アドレス有効→$\overline{\text{STERM}}$ アサート	t_{AVSL}	12-3
d	アドレス・ストローブのアサート→$\overline{\text{STERM}}$アサート	t_{SASL}	12-4
e	アドレス有効→$\overline{\text{BERR}}$/$\overline{\text{HALT}}$ アサート	t_{AVBHL}	12-5
f	アドレス・ストローブのアサート→$\overline{\text{BERR}}$/$\overline{\text{HALT}}$アサート	t_{SABHL}	12-6
g	アドレス有効→データ有効	t_{AVDV}	12-7
h	アドレス・ストローブのアサート→データ有効	t_{SADV}	12-8

図12-8　アクセス時間の計算図

さのバス・サイクルでは、t_{AVSL}(またはt_{SASL})とt_{AVDL}(またはt_{SADL})を比較すると、30nsの余裕時間があるため、$\overline{\text{DSACKx}}$より$\overline{\text{STERM}}$のタイミング条件の方が緩やかです。ただし、$\overline{\text{STERM}}$信号はバス・サイクル中はクロックのすべての立上りエッジに対して、それぞれ仕様#60および#61で規定されるセットアップおよびホールド時間を満足しなければならないという制約が加わります。表12-2で、合計クロック周期数(N) が2のときのt_{SASL}の値については、さらに説明する必要があります。

等式12-4を用いたアクセス時間の計算値は、ある条件のもとでは0になるため、すべての周波数において、ハードウェアが常に$\overline{\text{AS}}$で$\overline{\text{STERM}}$を認可できるとはかぎりません。しかし、このような条件付けはMC68030では不要です。$\overline{\text{STERM}}$は$\overline{\text{ECS}}$のアサート、S0の立上りエッジ、あるいは最も簡単にはアドレス・デコードまたはコンパレータ・ロジックの出力によっても発生させることができます。MC68030はバス・サイクルを開始し、$\overline{\text{AS}}$をアサートする前に、それらのバス・サイクルをアボートすることができるため、システムの他のデバイスがASによるアクセスの認可を必要とする場合もあります。

システムでCPUからメモリへのアクセス時間を最適化するための別の方法は、特定のMC68030デバイスの最大定格より低いクロック周波数を使用することです。表12-3に各種のクロック周波数でMC68030RC16およびMC68030RC20を動作させたときのt_{AVDV}(等式12-7)の計算結果を示しま

表12-2　20MHzでのメモリ・アクセス時間の等式

	N=2	N=3	N=4	N=5	N=6
(12-1) $t_{AVDL}=(N-1)\cdot t1-t2-t6-t47A$	—	46 ns	96 ns	146 ns	196 ns
(12-2) $t_{SADL}=(N-2)\cdot t1-t9-t47A$	—	26 ns	76 ns	126 ns	176 ns
(12-3) $t_{AVSL}=(N-1)\cdot t1-t6-t60$	21 ns	71 ns	121 ns	171 ns	221 ns
(12-4) $t_{SASL}=(N-1)\cdot t1-t3-t9-t60$	1 ns	51 ns	101 ns	151 ns	201 ns
(12-5) $t_{AVBHL}=N\cdot t1-t2-t6-t27A$	40 ns	90 ns	140 ns	190 ns	240 ns
(12-6) $t_{SABHL}=(N-1)\cdot t1-t9-t27A$	20 ns	70 ns	120 ns	170 ns	220 ns
(12-7) $t_{AVDV}=N\cdot t1-t2-t6-t27$	46 ns	96 ns	146 ns	196 ns	246 ns
(12-8) $t_{SADV}=(N-1)\cdot t1-t9-t27$	26 ns	76 ns	126 ns	176 ns	226 ns

ただし、
tX = AC電気的仕様＃Xを参照
t1 = クロック周期
t2 = クロック“L”時間
t3 = クロック“H”時間
t6 = クロック“H”からアドレス有効までの時間
t9 = クロック“L”から$\overline{AS}$“L”までの遅延
t27 = クロック“L”に対するデータ入力のセットアップ時間
t27A = クロック“L”に対する$\overline{BERR}/\overline{HALT}$のセットアップ時間
t47A = 非同期入力のセットアップ時間
t60 = クロック“H”に対する同期入力のセットアップ時間
N = バス・サイクル（ノン・バースト）の全クロック周期数
（同期サイクルではN ≧2、非同期サイクルではN≧3）

表12-3　CPUの最大定格周波数以下の周波数で動作させるためのt_{AVDV}の計算値

等式12-7のt_{AVDV}		MC68030RC20		MC68030RC25		
バス・サイクル(N)当たりのクロック数とその種類	ウエイトステート	16.67MHzのクロック	20MHzのクロック	16.67MHzのクロック	20MHzのクロック	25MHzのクロック
2クロック、同期	0	61	46	68	53	38 –
3クロック、同期	1	121	96	128	103	78
3クロック、非同期	0	121	96	128	103	78
4クロック、同期	2	181	146	188	153	1118
4クロック、非同期	1	181	146	188	153	118
5クロック、同期	3	241	196	248	203	158
5クロック、非同期	2	241	196	248	203	158
6クロック、同期	4	301	246	308	253	198
6クロック、非同期	3	301	246	308	253	198

す。システムが他のクロック周波数を使用した場合は、上記の等式を使用して正確なアクセス時間を計算することができます。

12.4.2　バースト・モード・サイクル

バースト・モード・バス・サイクルのメモリ・アクセス時間は、最初のアクセスのときにだけ上記の等式に従います。後続(2番目、3番目、そして4番目)のアクセスでは、メモリ・アクセス時間の計算は、バースト・モード・メモリ・システムのアーキテクチャに依存します。

アーキテクチャ上のトレードオフには、バースト・メモリ幅と使用するメモリの種類が含まれます。メモリ幅が128ビットの場合は、後続のオペランド・アクセスがクリティカル・タイミング・パスに影響を与えることはありません。たとえば、3-1-1-1のバーストが128ビット幅のメモリにアクセスする場合、最初のアクセスには表12-2にあるN=3の等式が適用されます。後続のアクセスも、これらの値を基準として使用しますが、さらにクロック周期が加算されます。2番目のアクセス

には1クロック周期、3番目のアクセスには2クロック周期、そして4番目のアクセスには3クロック周期が加算されます。このようにして、最初のサイクルのアクセス時間でクリティカル・タイミング・パスを決めます。

64ビット幅のメモリについては、アクセス時間と部品点数において、上記の2つの構成の間で妥協したものになります。

12.5 スタティックRAMメモリ・バンク

高クロック周波数で動作するMC68030では、ノー・ウエイト・ステートの外部メモリ・システムは、ほとんどの場合スタティックRAMで構成されているはずです。以下の項では、3つのスタティック・メモリ・バンクについて検討します。これらは、回路図どおり使用することができ、また外部キャッシュ設計にとりかかるときの参考にもなります。また、これらの設計は性能レベル、バス利用、およびコストの違いを反映したものです。

12.5.1 SRAMを使用した2クロック同期メモリ・バンク

MC68030は、一般に外部メモリ・システムが2クロック同期式バス・プロトコルをサポートできる場合に最高の性能を発揮します。この項では、2クロック・アクセスを使用して、20MHzのMC68030と一体となって動作する64Kバイトの完全なメモリ・バンクについて述べます。また、いくつかのオプション、およびコストや消費電力を低減するための簡単な変更についても検討します。図12-9に、完全なメモリ・バンクとそれをMC68030に接続する方法を示します。図に示すように、この回路には次の部品が必要です。

(8) 16K×4SRAM、アクセス・タイム35ns、独立したI/Oピン付き

(4) 74F244バッファ

(2) 74F32 ORゲート

(1) PAL16L8D(または相当品)

システムは必要に応じて(たとえば、複数の同期メモリ・バンクまたはポートが存在するため)、$\overline{\text{STERM}}$統合回路も用意しなければなりません。図12-9にこの統合回路をANDゲートとして示します。

メモリ・バンクは次の3つのセクションに分けることができます。

1. バイト選択およびアドレス・デコード・セクション(PALで供給)
2. 実メモリ・セクション(SRAM)
3. バッファ・セクション

最初のセクションはORゲート74F32 2個、D型のフリップ・フロップ74F74 1個およびPAL16L8D 1個で構成されています。図12-10にPALの等式の例を示します。PALは6つのマップト信号を発生します。つまり、ライト操作のための4つのバイト選択信号、1つのバッファ制御信号、そして1つのサイクル終了信号です。バイト選択信号は、プロセッサがメモリ・バンク内の64Kバイトをアドレス指定している間、適当な単数または複数のバイトがSIZ0,SIZ1,A0およびA1信号に従って、書き込まれているときにだけアサートされます。$\overline{\text{UUCS}}$、$\overline{\text{UMCS}}$、$\overline{\text{LMCS}}$、および$\overline{\text{LLCS}}$の4つの信号は、それぞれデータ・ビットD24-D31、D16-D23、D8-D15、およびD0-D7のデータ・ビットを制御します。アドレスが有効になる前に、メモリに誤って書込みが行なわれないようにするため、$\overline{\text{AS}}$をバイト選択信号の条件として使用しています。リード操作中には、$\overline{\text{AS}}$で有効となったリード・チップ選択($\overline{\text{RDCS}}$)信号は、データ・バッファだけを制御します(メモリはすでに、$\overline{\text{E}}$入力を接地してイネーブルされているため)。最後にPALは$\overline{\text{TERM}}$信号を発生します。

等式に示すとおり、$\overline{\text{TERM}}$は2つのイベントからなります。1つはリード・サイクルに対応し、も

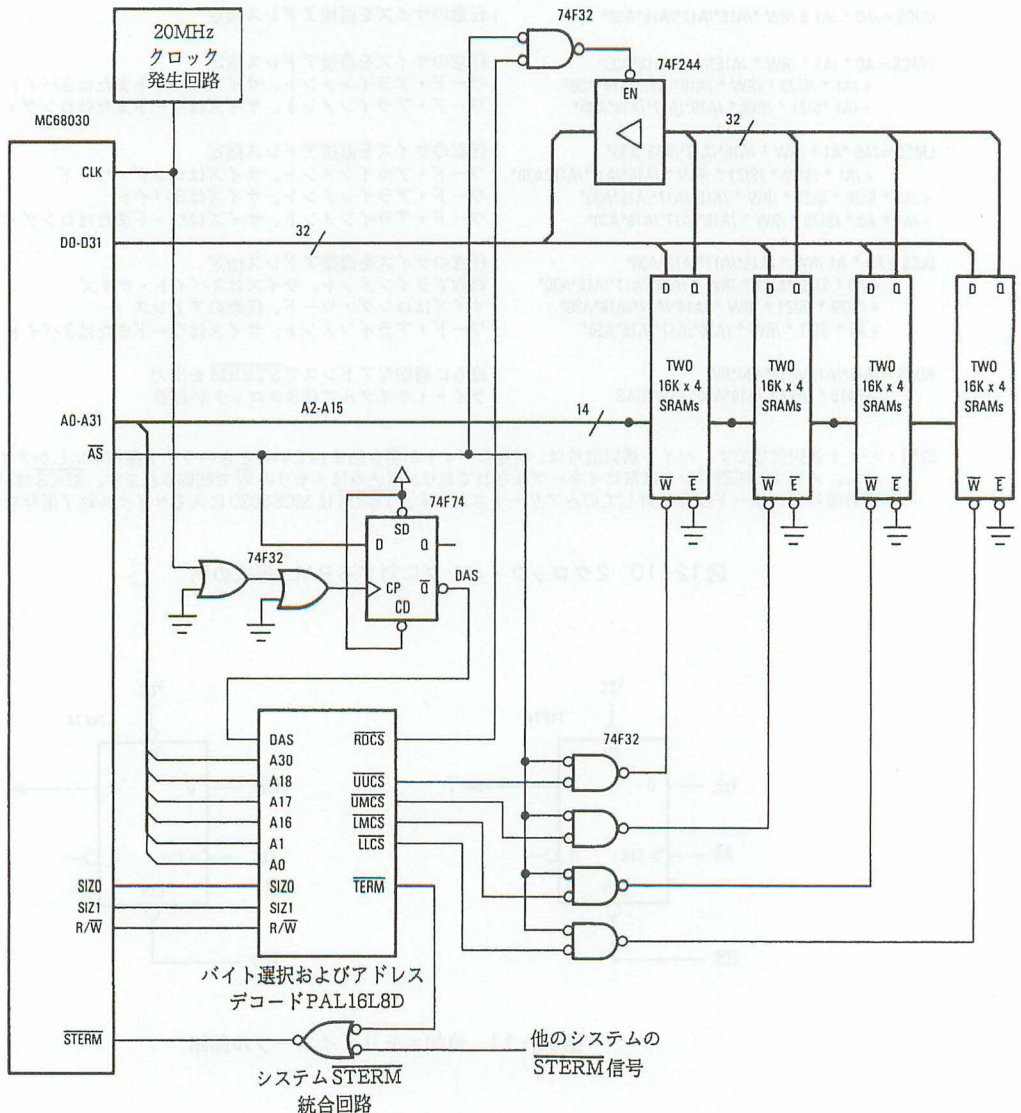



図12-9　2クロック・リード、3クロック・ライトのメモリ・バンクの例

う1つはライト・サイクルに対応しています。リード・サイクルでは、TERMは、アドレスがエンコードされたSRAMのメモリ・マップト・バンクに一致するたびにアサートされるアドレス・デコード信号です。ライト操作では、同じアドレス・デコードを認可するのにAS(DAS)を遅延させた信号を使用し、ライト操作を3クロック・サイクルまで延長します。DAS信号は、クロック信号を2個の74F32 ORゲートを通過させてから74F74のD型フリップ・フロップに接続することによって、クロック・エッジから遅れて発生します。これによって、TERM信号を発生させるための最大遅延時間がMC68030の同期入力のホールド時間に違反しないことが保証されます。MC68030は、ライト操作を3クロック・サイクルまで延長することによって、ライト・ストローブ(W)がネゲートされるまでに、指定されたデータ・セットアップ時間を容易にSRAMに適合させることができます。これで、TERMはシステムのSTERM統合回路に接続されます。統合回路の伝搬遅延時間は15ns以下

```
UUCS=/A0 * /A1 8 /RW */A16*/A17*/A18*A30*            ;任意のサイズを直接アドレス指定

UMCS=A0 * /A1 * /RW * /A16*/A17*/A18*A30*            ;任意のサイズを直接アドレス指定
     +/A1 * /SIZ0 * /RW * /A16*/A17*/A18*A30*        ;ワード・アラインメント、サイズはバイトまたは3バイト
     +/A1 *SIZ1 * /RW * /A16*/A17*/A18*A30*          ;ワード・アラインメント、サイズはワードまたはロング・ワード

LMCS=/A0 *A1 * /RW * /A16*/A17*/A18*A30*             ;任意のサイズを直接アドレス指定
     +/A1 * /SIZ0 * /SIZ1 * /RW * /A16*/A17*/A18*A30* ;ワード・アラインメント、サイズはロング・ワード
+/A1 * SIZ0 * SIZ1 * /RW * /A16*/A17*/A18*A30*       ;ワード・アラインメント、サイズは3バイト
+/A1 * A0 * /SIZ0 * /RW * /A16*/A17*/A18*A30*        ;ワード・アラインメント、サイズはワードまたはロング・ワード

LLCS=A0 * A1 /RW * /A16*/A17*/A18*A30*               ;任意のサイズを直接アドレス指定
     +A0 * SIZ0 *SIZ1 * /RW * /A16*/A17*/A18*A30*    ;奇数アラインメント、サイズは3バイト・サイズ
     +/SIZ0 * /SIZ1 * /RW * /A16*/A17*/A18*A30*      ;サイズはロング・ワード、任意のアドレス
     +A1 * SIZ1 * /RW * /A16*/A17*/A18*A30*          ;ワード・アラインメント、サイズはワードまたは3バイト・サイズ

RDCS=/A16*/A17*/A18*A30*RW                           ;直ちに適切なアドレスでSTERMを出力
     +/A15 * /A17 * /A18*A30* /RW*DAS                ;ライト・サイクルには3クロックが必要
```

説明：バイト選択信号です。バイト選択信号は、特定のバイトが書き込まれているときのライト操作中にしかアサートされません。メモリの同期バンクは常にイネーブルされており、書込みはメモリの$\overline{W}$で制御されます。$\overline{RDCS}$はバッファ制御用の信号で、リード操作に対してのみアサートされます。$\overline{TERM}$はMC68030に入るサイクル終了信号です。

図12-10　2クロック・バンクに対するPAL等式の例

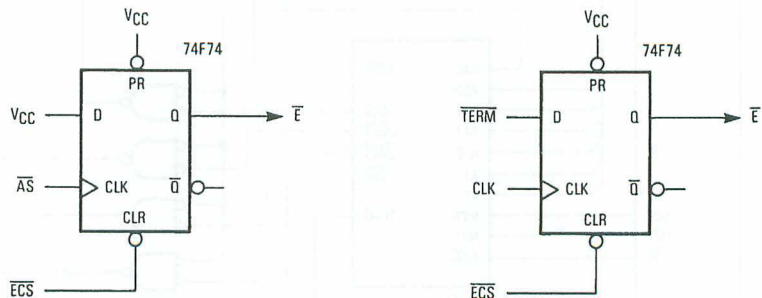



図12-11　追加メモリ・イネーブル回路

でなければなりません。システムに他の同期メモリまたはポートがなければ、$\overline{TERM}$は直接$\overline{STERM}$に接続することができます。

2番目のセクションにはメモリ・デバイスが含まれています。ここでは8個のデバイスを使用していますが、この数を増やしてEDACをサポートしたり、密度が高くなるように設計することができます。この設計で使用するメモリ・デバイスの最も重要な特長は、データ・インとデータ・アウトのピンが別々になっていることであり、これによってアドレス・デコードが完了する前に、データ・バスを競合させることなく、SRAMをイネーブルできます。SRAMのイネーブル・ピンは、回路を簡素化するため、そしてメモリ・アクセスのタイミングを改善するために接地されています。設計者が、バスの利用率を下げて消費電力を抑えるために、何らかのイネーブル回路を組み入れたい場合、メモリの$\overline{E}$信号をステートS0の立下りエッジより前(アドレスが有効になると同時またはその前)にアサートすれば、この設計のタイミングが維持されます。考えられる2つのイネーブル回路を図12-11に示します。

メモリ・バンクの3番目のセクションはデータ・バッファです。データ・バッファは74F244となっていますが、74AS244を使用することもできます。前述のとおり、リード操作中はASで認可さ

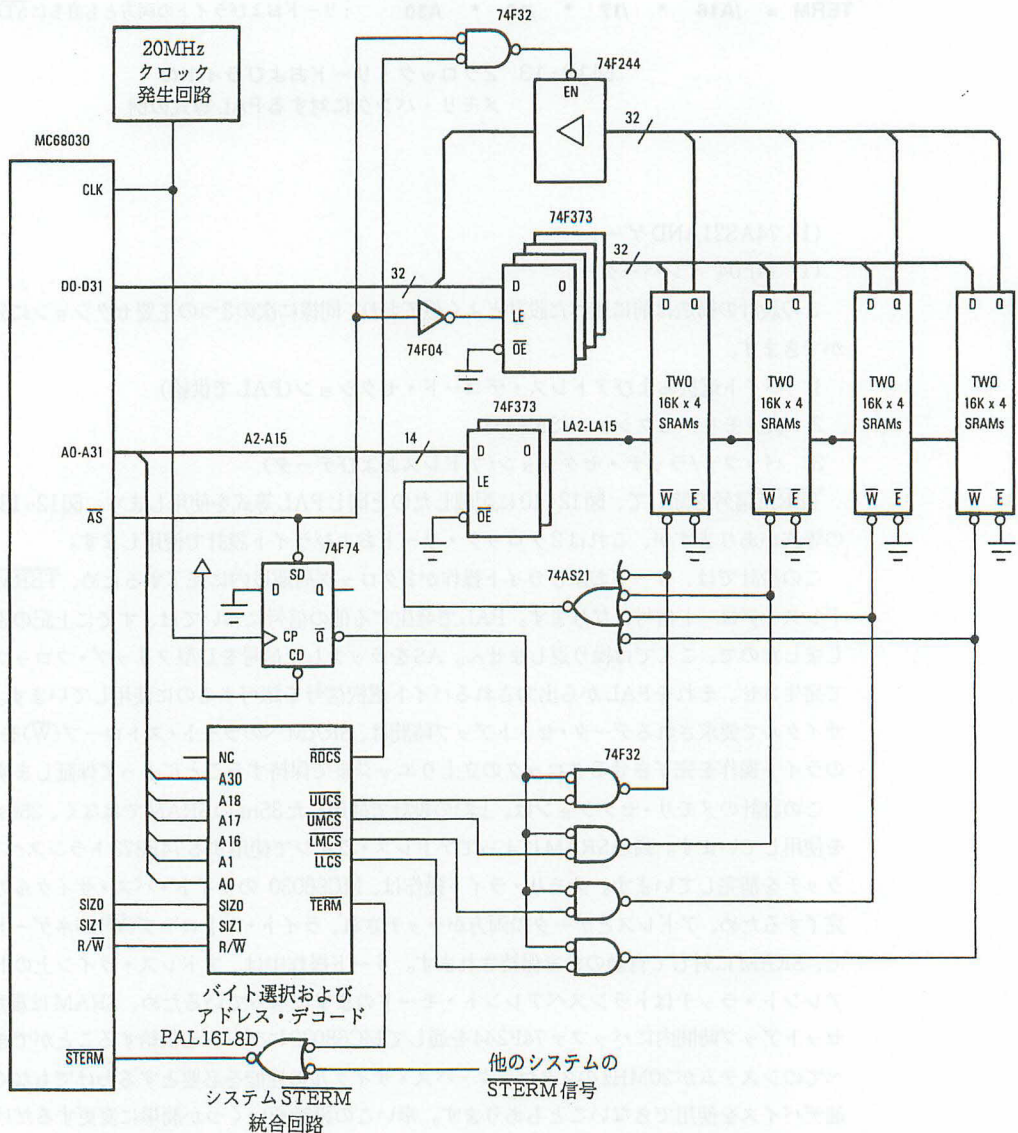



図12-12　2クロック・リードおよびライトのメモリ・バンクの例

れたRDCS信号でデータ・バッファを制御します。

性能を最大限に高めるのに、リードおよびライト操作の両方とも2クロック・サイクル以内に完了しなければなりません。図12-12に2クロック・リードおよびライトのメモリ・バンクを示します。この回路に必要な部品は次のとおりです。

(8) 16K×4SRAM、アクセス・タイム25ns、独立したI/Oピン付き

(4) 74F244 バッファ

(2) 74F32 ORゲート

(1) PAL16L8D(または同等品)

(1) 74F74 D型フリップ・フロップ

(2) 74F373 トランスペアレント・ラッチ

TERM = /A16 * /17 * /16 * A30 ;リードおよびライトの両方とも直ちに$\overline{\text{STERM}}$を発生

図12-13 2クロック・リードおよびライトの
メモリ・バンクに対するPAL等式の例

(1) 74AS21ANDゲート

(1) 74F04 インバータ

この設計の構造は前に述べた設計とよく似ており、同様に次の3つの主要セクションに分けることができます。

1. バイト選択およびアドレス・デコード・セクション(PALで供給)
2. 実メモリ・セクション(SRAM)
3. バッファ/ラッチ・セクション(アドレスおよびデータ)

$\overline{\text{TERM}}$信号を除いて、図12-10に記載したのと同じPAL等式を使用します。図12-13に$\overline{\text{TERM}}$の等式がありますが、これは2クロック・リードおよびライト設計で使用します。

この設計では、リードおよびライト操作が2クロック周期以内に完了するため、$\overline{\text{TERM}}$は単にアドレス・デコード信号になります。PALで発生する他の信号については、すでに上記の設計で説明しましたので、ここでは繰り返しません。ASをラッチした信号をD型フリップ・フロップの74F74で発生させ、それをPALから出力されるバイト選択信号を認可するのに使用しています。ライト・サイクルで要求されるデータ・セットアップ時間は、SRAMへのライト・ストローブ($\overline{\text{W}}$)をMC68030のライト操作を完了させるクロックの立上りエッジまで保持することによって保証します。

この設計のメモリ・セクションは、上記の設計で使用した35nsのSRAMではなく、25nsのSRAMを使用しています。高速SRAMによってアドレス・ラインで使用する74F373トランスペアレント・ラッチを補完しています。メモリ・ライト操作は、MC68030のライト・バス・サイクルの終了後に完了するため、アドレスとデータの両方がラッチされ、ライト・ストローブ($\overline{\text{W}}$)がネゲートされるまで、SRAMに対して有効のまま保持されます。リード操作中は、アドレス・ライン上のトランスペアレント・ラッチはトランスペアレント・モードのままになっているため、SRAMは規定データ・セットアップ時間内にバッファ74F244を通してMC68030にデータを供給することができます。すべてのシステムが20MHzの2クロック・バス・サイクルの性能を必要とするわけでもなく、また高速デバイスを使用できないこともあります。幸いこの設計をいくつか簡単に変更するだけで、さまざまな価格対性能比を実現できます。

最も簡単かつ直接的な方法は、MC68030のクロック周波数を低くすることです。たとえば、クロック周波数が約18.1MHz 以下になれば、同じ制御ロジックは45ns(クロック周波数が15.8MHz 以下の場合は55ns)メモリで2クロック・バス・サイクルをサポートします。クロック周波数を20MHzにしなければならない場合は、3クロック・バス・サイクルで動作させることができます。これは、フリップ・フロップを1個追加して、$\overline{\text{TERM}}$信号を1クロック遅らせることによって可能です。その結果、メモリ・アクセス時間は、3クロック・バス・サイクルで動作する20MHzのプロセッサの場合、85ns以上になります。

12.5.2 SRAMを用いた2-1-1-1 バースト・モード・メモリ・バンク

MC68030は外部メモリ・システムが2-1-1-1 のバースト・プロトコルをサポートできれば、最もバスの利用が少なくてすみます。ただし、これには例外があります。たとえば、多くのメモリ・アクセスが参照の局在性の原理に基づいて行なわれていない場合は、バースト・アクセスでバスの利用を低減することはできません。この項では、20MHzのMC68030で動作可能な256Kバイトの完全

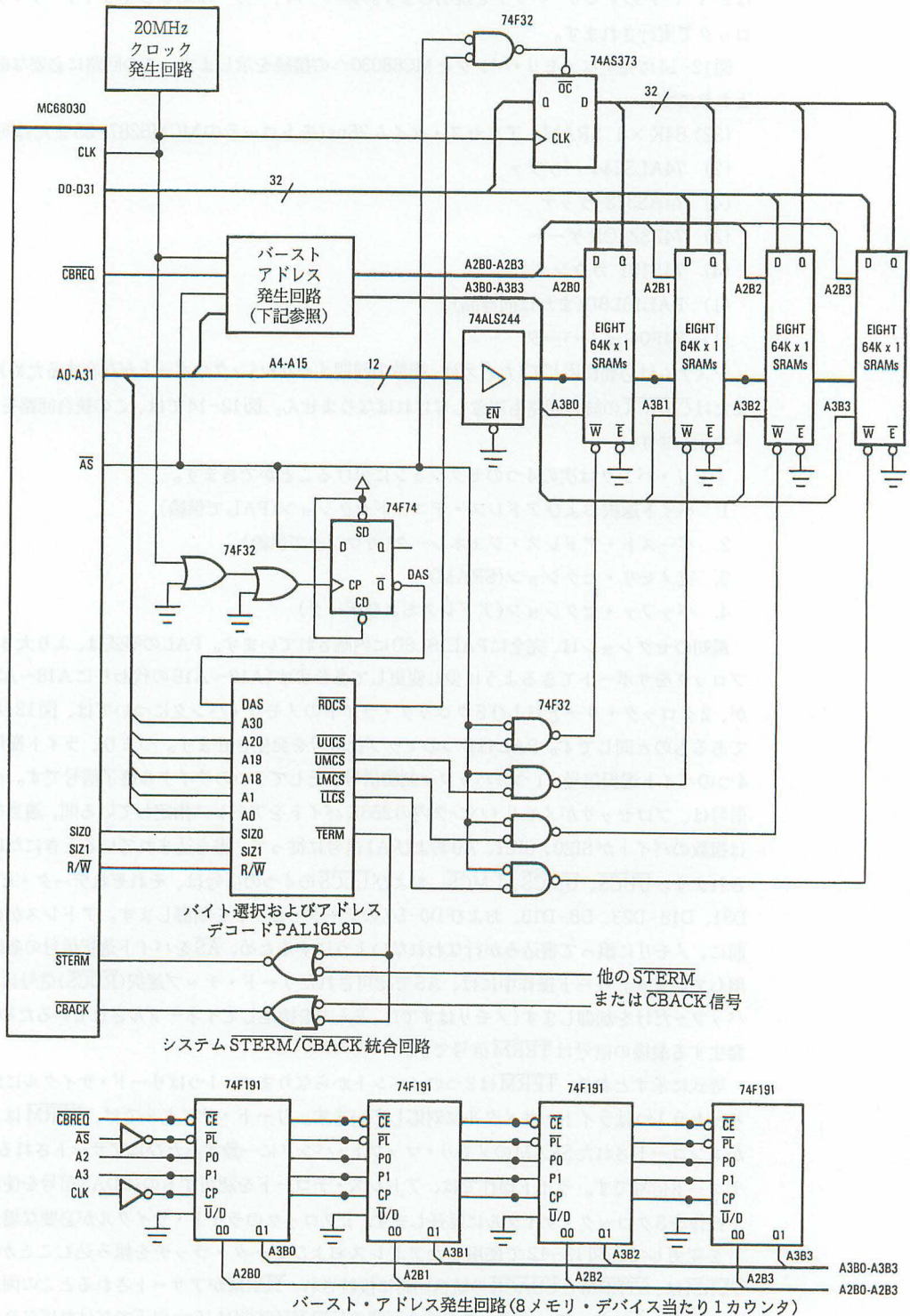



図12-14　20MHz、256Kバイトでの2-1-1-1 バースト・モードのメモリ・バンクの例

な2-1-1-1 のメモリ・バンクを説明します。非バースト・リードおよび全ライト・サイクルは2クロックで実行されます。

図12-14に完全なメモリ・バンクとMC68030への接続を示します。この回路に必要な部品は次のとおりです。

(32) 64K × 1 SRAM、アクセス・タイム25ns(モトローラのMCM6287-25または同等品)
(2) 74ALS244 バッファ
(4) 74AS373 ラッチ
(2) 74F32 ORゲート
(4) 74F191 カウンタ
(1) PAL16L8D(または同等品)
(1) 74F04 インバータ

システムは必要に応じて(たとえば、複数の同期メモリ・バンクやポートが存在するため)、$\overline{\text{STERM}}$または$\overline{\text{CBACK}}$の統合回路も用意しなければなりません。図12-14では、この統合回路をANDゲートで示します。

メモリ・バンクは次の4つのセクションに分けることができます。

1. バイト選択およびアドレス・デコードセクション(PALで供給)
2. バースト・アドレス・ジェネレータ(カウンタで供給)
3. 実メモリ・セクション(SRAM)
4. バッファ・セクション(アドレスおよびデータ)

最初のセクションは、完全にPAL16L8Dに内蔵されています。PALの等式は、より大きいメモリ・ブロックをサポートできるように少し変更してあります(A16～A18の代わりにA18～A20を使用)が、2クロック・リードおよび3クロック・ライトのメモリ・バンクについては、図12-8に掲載してあるものと同じです。PALは6つのマップ化信号を発生させます。つまり、ライト操作のための4つのバイト選択信号、1つのバッファ制御信号、そして1つのサイクル終了信号です。バイト選択信号は、プロセッサがメモリ・バンク内の256Kバイトをアドレス指定している間、適当な単数または複数のバイトがSIZ0、SIZ1、A0およびA1信号に従って、書き込まれているときにだけアサートされます。$\overline{\text{UUCS}}$、$\overline{\text{UMCS}}$、$\overline{\text{LMCS}}$、および$\overline{\text{LLCS}}$の4つの信号は、それぞれデータ・ビットD24-D31、D16-D23、D8-D15、およびD0-D7のデータ・ビットを制御します。アドレスが有効になる前に、メモリに誤って書込みが行なわれないようにするため、$\overline{\text{AS}}$をバイト選択信号の条件として使用しています。リード操作中には、$\overline{\text{AS}}$で認可されたリード・チップ選択($\overline{\text{RDCS}}$)信号は、データ・バッファだけを制御します(メモリはすでに、$\overline{\text{E}}$入力を接地してイネーブルされているため)。PALが発生する最後の信号は$\overline{\text{TERM}}$信号です。

等式に示すとおり、$\overline{\text{TERM}}$は2つのイベントからなります。1つはリード・サイクルに対応しており、もう1つはライト・サイクルに対応しています。リード・サイクルでは、$\overline{\text{TERM}}$は、アドレスがエンコードされたSRAMのメモリ・マップト・バンクに一致するたびにアサートされるアドレス・デコード信号です。ライト操作では、アドレス・デコードを認可するのにDAS信号を使用し、ライト操作を3クロック・サイクルに延長します。2クロックのライト・サイクルが必要な場合、この設計を変更して、図12-12で使用したアドレスおよびデータ・ラッチを組み込むことができます。$\overline{\text{TERM}}$は、$\overline{\text{STERM}}$と$\overline{\text{CBACK}}$の統合回路に接続され、$\overline{\text{TERM}}$がアサートされるとこの両方がアサートされるようになっています。この統合回路の伝搬遅延時間は15ns以下でなければなりません。システムに他の同期メモリまたはポートがなければ、$\overline{\text{TERM}}$を直接$\overline{\text{STERM}}$に接続し、$\overline{\text{CBACK}}$を接地することができます。

2番目のセクションはバースト・アドレス・ジェネレータで、4個のカウンタとインバータで構成されます。これらのカウンタは、MC68030のアドレス・ライン(A2およびA3)をバッファするため、

およびバースト動作中に次のロング・ワード・アドレスを供給する役割があります。74F191は$\overline{AS}$がネゲートされているときは、各バス・サイクルの初めで非同期にプリセットされます。$\overline{AS}$がアサートされているときには、カウント動作は$\overline{CBREQ}$信号およびCLK信号に依存します。ライト操作中には、$\overline{CBREQ}$は常にネゲートされており、カウンタはアドレス・バッファとしてしか機能しません。リード操作中に$\overline{CBREQ}$がアサートされると、カウンタ・ビットQ1：Q0の現在の値は、$\overline{AS}$がアサートされた後、MC68030のクロックの立下りクロック・エッジでインクリメントされます。バッファの伝搬遅延が大きくならないようにし、十分なドライブ能力を得るために、4個のカウンタを使用しています。各カウンタは8個のメモリ・デバイスをドライブします。

3番目のセクションにはメモリ・デバイスがあります。この設計で使用するメモリ・デバイスの最も重要な特長は、データ・インとデータ・アウトのピンが別になっていることであり、これによってアドレス・デコードが完了する前に、データ・バスの競合を招くことなく、SRAMをイネーブルしたままにしておくことができます。設計者が、バスの利用率を下げるために、何らかのイネーブル回路を組み入れたい場合、メモリの$\overline{E}$信号をステートS0の立下りエッジ後13ns以内にアサートすれば、この設計のタイミングが確保されます。

メモリ・バンクの4番目、最後のセクションはアドレスおよびデータ・バッファです。アドレス・バッファは74ALS244となっていますが、74AS244および74F244を使用することもできます。アドレス・バッファへの2入力は、未使用のままになっていますので、適当な記憶密度のSRAMが使用できるようになったときに、デバイスを追加しないでも、1Mバイトまで拡張できます。$\overline{AS}$で認可した$\overline{RDCS}$信号でリード操作中のデータ・バッファを制御します。アドレス・バッファは常にイネーブルされます。

この設計をいくつか変更すれば性能を改善することができます。特に$\overline{CBACK}$を制御して、バースト・サイクルを禁止あるいは中断する回路は簡単に追加できます。この回路には2つの機能がなければなりません。その1つはラップ・アラウンドを防止する機能で、もう1つはデータ・オペランドがロング・ワード境界にまたがる場合にバースト動作を防止する機能です。

すべてのシステムが20MHzの2クロック・バス・サイクルの性能を必要とするわけでもなく、また高速デバイスを使用できないこともあります。幸いこの設計をいくつか簡単に変更するだけで、さまざまな価格対性能比を実現できます。

すべてのシステムが20MHzの2-1-1-1 バースト・サイクルの性能を必要とするわけではなく、また設計に高速デバイスを使用できないこともあります。クロック周波数が約17.5MHz 以下であれば、同じサポート・ロジックが35nsメモリで2-1-1-1 バースト・サイクルをサポートします。それでもなお、20MHzの周波数を選択する場合、設計者は3-1-1-1 バースト・サイクル動作を選択することができます。

12.5.3　SRAMを使用した3-1-1-1 バースト・モード・メモリ・バンク

図12-15に20MHzのMC68030とともに動作可能な、256Kバイトの完全な3-1-1-1メモリ・バンクを示します。この回路に必要な部品は次のとおりです。

(32) 64K ×1SRAM 、アクセス・タイム35ns(モトローラのMCM6287-35または同等品)
(4)　74ALS244 バッファ
(4)　74F374 ラッチ
(2)　74F32 ORゲート
(4)　74F191 カウンタ
(1)　PAL16L8D(または同等品)
(2)　インバータ
(1)　フリップ・フロップ

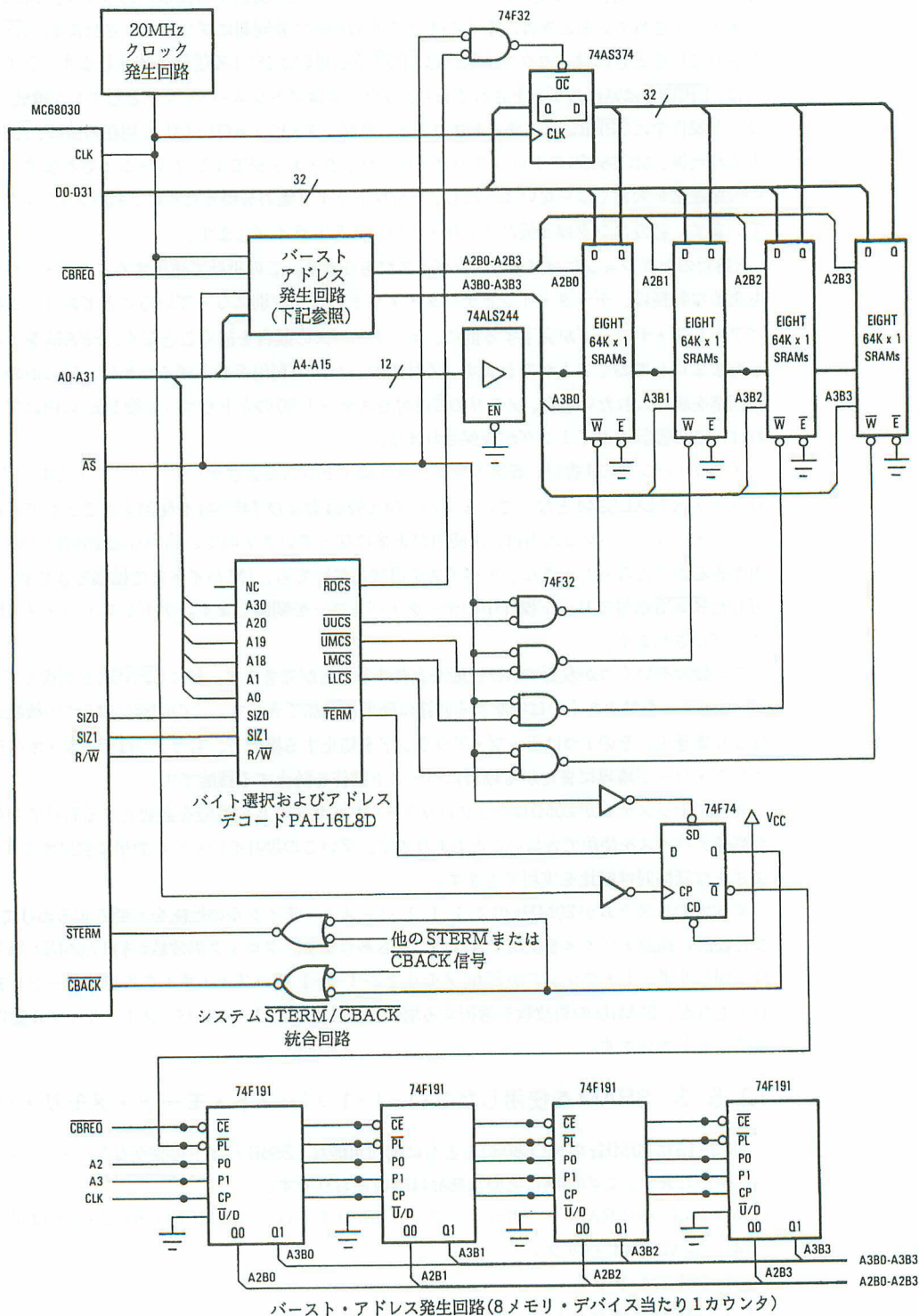



図12-15 20MHz、256Kバイトでの3-1-1-1パイプライン・バースト・モードのメモリ・バンクの例

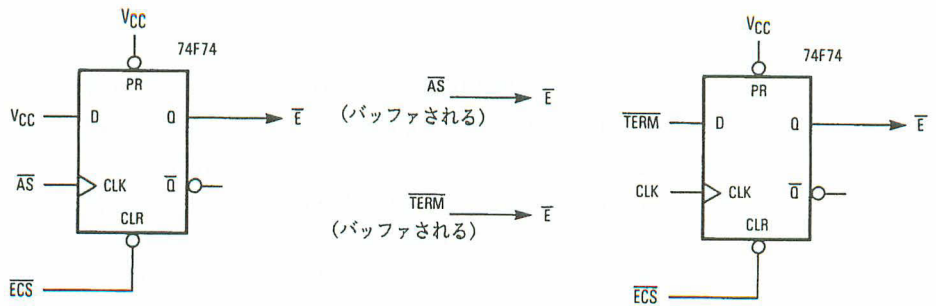



図12-16　追加メモリ・イネーブル回路

このメモリ・バンクの構造は、「12. 5. 2　SRAMを用いた2-1-1-1 バースト・モード・メモリ・バンク」で記載した2-1-1-1 メモリ・バンクと非常によく似ています。実際、PALとアドレス・バッファはまったく同じです。PAL の等式は図12-10 に示します。最も重要な違いは、データのラッチ方法であり、ここではフリップ・フロップを使用しています。また、D型のフリップ・フロップをPALの入力側から$\overline{\text{TERM}}$出力側に移動しています。

データ・フリップ・フロップにより、メモリから取り出したロング・ワードを、容易にセットアップ時間およびホールド時間を満足させながらパイプ・ラインに入れることができます。MC68030が“現在の”ロング・ワードをラッチする前でも、メモリ・デバイスは次のロング・ワードのデータを発生しています。この変換によって、アクセスのタイミング条件が簡単になるため、20MHzのクロック周波数で35nsのメモリを使用できるようになります。クロック周波数が17MHz以下の場合は、45nsのメモリを使用できます。遅いサイクルを使用するもう1つの利点は、SRAMのイネーブル入力のタイミング条件が緩和されることです。図12-15ではすべてのSRAMチップ・イネーブルが接地されていますが、メモリの$\overline{\text{E}}$信号がステートS2の立上りエッジから10ns以内にアサートされれば、引き続きこの設計のタイミングを使用できます。図12-16に考えられる4つのイネーブル回路を示します。

$\overline{\text{TERM}}$信号に接続されるフリップ・フロップは、2つの役割を果たしています。その1つは、最初のロング・ワードにウエイト・ステートを挿入するために、$\overline{\text{TERM}}$信号をサイクルの初めに遅らせることです。もう1つは、74F374が最初のロング・ワードをラッチするまで、バースト・アドレス・ジェネレータがロング・ワードのベース・アドレスをインクリメントしないようにすることです。

2-1-1-1の設計で説明した性能強化のための変更も同じくこの設計に適用されます。特に$\overline{\text{CBACK}}$を制御してバースト・サイクルを禁止または中断する回路を追加することができます。この回路には2つの機能がなければなりません。その1つはラップ・アラウンドを防止する機能で、もう1つはデータ・オペランドがロング・ワード境界にまたがる場合にバースト動作を防止する機能です。もう1つの機能強化としては、$\overline{\text{TERM}}$制御回路を変更してライト・ラッチ機構を追加し、2クロック・ライトを実行できるようにすることです。

3-1-1-1 メモリ・バンクのクリティカル・パスは、2-1-1-1 メモリ・バンクのように最初のロング・ワード・アクセスではなく、バースト・サイクルでは後続のロング・ワードになります。2-1-1-1 バースト・サイクルのクリティカル・パスを修正できるアーキテクチャは1つしかありません。ただし、設計者は3-1-1-1 バースト・サイクルに対し、64 ビットまたは128 ビット幅のメモリ・バンクを形成することができます。

この方法では後続のロング・ワードのアクセス時間を前のロング・ワードのアクセスの中に隠すことができます。

12. 6　外部キャッシュ

メモリへの平均アクセス時間を低くするために、一部のシステムは最も近い時点で使用した命令またはデータ、あるいはその両方を格納するメイン・プロセッサに対してローカルなキャッシュを実現することができます。MC68030では、キャッシュの設計者にはいくつかのアーキテクチャ上のオプションが用意されています。最も重要な決定は、キャッシュを同期式デバイスとして構成するか非同期デバイスとするかということ、そしてキャッシュ・アクセスをアーリ・ターミネーション(キャッシュのルックアップが完了する前)とするか、妥当性の検査のあとに終了させるかということです。

MC68030のレイト$\overline{BERR}$/$\overline{HALT}$機能によって、外部デバイスは$\overline{DSACKx}$または$\overline{STERM}$をアサートし、エラー状態を検出したあとで(それぞれ、約1クロック周期または1.5クロック)そのサイクルをアボートするか再試行することによって、バス・サイクルの完了を知らせます。多くのメモリ構造の1つのクリティカル・アクセス・パスは、$\overline{DSACKx}$/$\overline{STERM}$をアサートして、ウエイト・ステートがさらに追加されるのを回避するため、レイト・アボート機能によって、メモリ・コントローラはプロセッサ・データ・バス上でデータが有効になる前にバス・サイクルを終了します。データの妥当性検査が失敗すれば、メモリはそのサイクルをアボート($\overline{BERR}$)または再試行($\overline{BERR}$/$\overline{HALT}$)することができます。この手法はメモリ・エラー検出回路において有用です。このメモリ・エラー検出回路では、データが使用可能になるとすぐにサイクルを終了することができ、またプロセッサがサイクルの終了を通知してからレイト再試行によってデータをラッチするまで、またはエラー表示によってアボートが通知されるまでの期間にエラー・チェックを行なうことができます。同様に、この手法は、キャッシュ・タグの妥当性検査が、サイクルの終了を通知しなければならない時点より前に完了することはできず、レイト・アボートまたレイト再試行を表示しなければならない時点より前には完了するキャッシュ機構で使用することができます。

外部キャッシュ・ミスに対し、レイト・サイクルの再試行を利用するかどうかを選択する上で主な検討事項は、キャッシュ・ミスが発生したあとのバス・サイクルの再試行に関係しています。最小のペナルティは、バス制御ストローブ($\overline{BERR}$および$\overline{HALT}$)がアボートされたサイクルの完了後、次のサイクルをすぐに開始できる間隔でネゲートされたと仮定すれば、そのサイクルを再試行するのに必要な4クロック周期(ミスが検出された2クロックと2クロックのアイドル・バス時間)です。このオーバヘッドを評価して、予定されたキャッシュ・ミス・レートによって、再試行しなければならないサイクルの割合を決めます。さらに、システムにおける並列化の程度も考慮しなければなりません。キャッシュ・ミスが発生したあと、プロセッサがそのサイクルを再試行している間に、メイン・メモリに対するバス・サイクルを継続することができ、レイト再試行に関連する制御ペナルティの一部またはすべてを回避することが可能です(それに必要な制御回路は、より複雑になるかもしれませんが)。

2クロック・バスまたはバースト機能に対しては、必ず同期バスを使用しなければなりませんが、3クロック以上の非バースト・キャッシュに対しては、同期操作または非同期操作のいずれかを選択する必要があります。妥当性検査のあとにしかバス・サイクルが終了しない場合は、同じ長さのバス・サイクルに対しては、アドレス有効から$\overline{STERM}$のアサートまでのタイミング条件のほうが、アドレス有効から$\overline{DSACK}$のアサートまでのタイミング条件よりも長いため、同期バスの使用が推奨されます。キャッシュがレイト・サイクル再試行をインプリメントしている場合は、どのバス制御モードを使用するかの選択はそれほど重要ではなく、システム特有の機能および制御構造によって決めます。外部キャッシュの中には、同期および非同期転送の両方を使用するものもあります。ヒットに対しては同期、ミスに対しては非同期、あるいはその逆というわけです。以下に述べる説明では、外部キャッシュが同期2クロック・プロトコルを使用しているものと仮定していますが、大部分の説

明は非同期プロトコルにも適用されます。

MC68030のMMUがディセーブルされると、すべてのバス・サイクルは論理アドレスを使用します。MMUがイネーブルされている場合は、外部アドレス・バスは物理アドレス(トランスペアレント変換(TTx)レジスタから直接マップされる論理-物理アドレスを含む)を使用します。論理および物理の2つの動作モードは、外部キャッシュのメンテナンスに影響を与えます。たとえば、外部キャッシュが物理アドレスを使用しているときには、コンテクスト・スイッチのたびにキャッシュをフラッシュする必要はありません。システムの各タスクは、論理アドレス空間の独自のマッピングをもっているため、論理キャッシュはシステムの論理-物理マッピングが変更されるたびに(コンテクスト・スイッチ中に発生する)、すべてのエントリをフラッシュしなければなりません。物理アドレス空間は1つしかなく、物理キャッシュでは特定のオペランドに対するすべての参照が同じ物理アドレスを使用していなければならないため、この問題は発生しません。

複数のタスクでキャッシュ・エントリを維持させようとするユーティリティを考慮するときには、意図するキャッシュ・サイズを計算しなければなりません。キャッシュが比較的小さく、コンテクスト・スイッチの間隔が長い場合、各タスクはキャッシュの充てんを行ない、前のタスクの実行中に生成されたすべてのエントリを取り除く傾向があります。逆に、キャッシュのサイズが比較的大きく、コンテクスト・スイッチの間隔が短い場合、キャッシュは効率よくエントリを共用していることになります。

12. 6. 1　キャッシュ・インプリメンテーション

外部キャッシュの構成例を図12-15に示します。この構成では、キャッシュのタイミング・コントローラは、キャッシュがアクセスを“ヒット”または“ミス”として確認するのに十分な時間が得られるまで、そのバス・サイクルを終了しません。“ヒット”と判断すると、キャッシュ・コントローラは$\overline{\text{STERM}}$信号をアサートし、$\overline{\text{AS}}$(A)が外部システムに伝達されるのを阻止します。MC68030が正常に$\overline{\text{AS}}$をアサートする前にキャッシュの決定を下すことができない場合は、その決定が有効になるまで、$\overline{\text{AS}}$の伝搬を遅らせるための機構が必要です。そうしないと、$\overline{\text{AS}}$信号が誤ってアサートされるおそれがあります。

キャッシュ制御回路(B)には、キャッシュのエントリをクリアまたは生成するのに必要な全ロジックが内蔵されています。また、(B)にはヒットまたはミスのいずれが発生したかを判断するのに必要な決定ロジック、そしてルックアップ回路および比較回路に有効な決定を下すのに十分な時間を与えるまで“ヒット”信号の伝搬を阻止するためのタイミング・ロジックが含まれています。このキャッシュの設計におけるクリティカル・パスは、MC68030が有効アドレスを出力してからキャッシュ・コントローラが$\overline{\text{STERM}}$をアサートするまでです(等式12-3)。キャッシュ・ヒットの決定が行なわれたあと、ヒット信号が直接$\overline{\text{STERM}}$信号をドライブします。$\overline{\text{AS}}$がアサートされたときに、適切なセットアップ時間およびホールド時間が守られていると考えられるときは、$\overline{\text{AS}}$で$\overline{\text{STERM}}$を認可する必要はありません。ノー・ウエイト・ステートにより20MHzで動作している場合、MC68030が有効アドレスを出力してからキャッシュ・コントローラが$\overline{\text{STERM}}$をアサートするまでに、21nsの余裕があり、またプロセッサでは有効アドレスからデータ有効までに46nsの余裕があります。

独特なキャッシュ・アーキテクチャ、サイズ、コスト、その他の理由により、アクセス時間を満足することができない場合、システム設計者は前述したアーリ・ターミネーション・アプローチの利用を選択することができます。アーリ・ターミネーションは、アドレス有効から$\overline{\text{BERR}}$/$\overline{\text{HALT}}$がアサートされるまでのクリティカル・パスを満足させることによって、キャッシュ・コントローラが利用できる決定時間を長くします(等式12-5)。図12-17に示すキャッシュ構造に必要な変更は、$\overline{\text{STERM}}$の生成だけです。図12-18に、MC68030と外部キャッシュ間に配置して、アーリ・ターミネーションまたはレイト再試行機能を提供する回路例を示します。

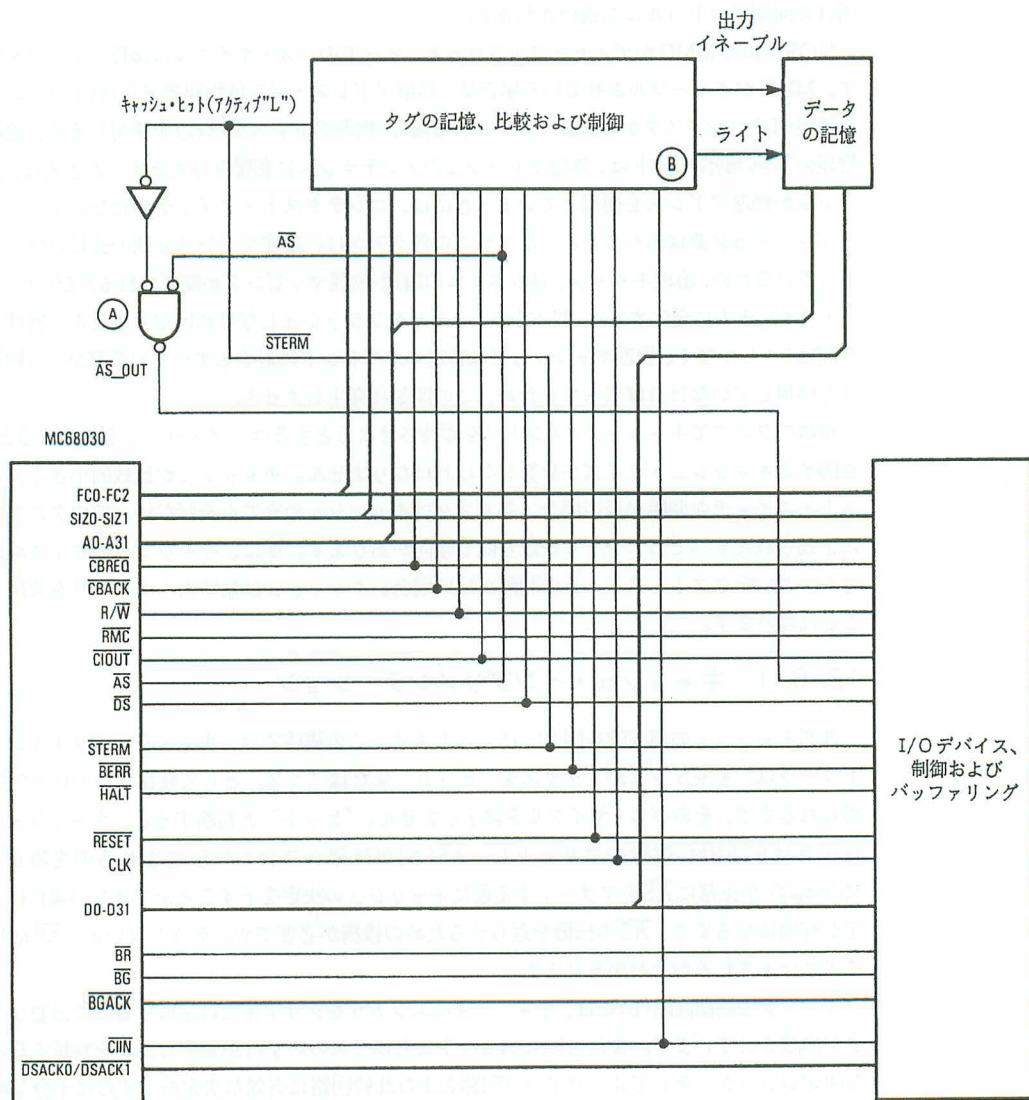



図12-17 外部物理キャッシュ付きMC68030 ハードウェア構成例

サイクルの後半でキャッシュが有効な“ヒット”を発生できると仮定すれば、通常$\overline{AS}$がアサートされるとすぐに回路(C)は$\overline{STERM}$信号をアサートして、バス・サイクルを終了します。回路(C) はまた、キャッシュ不可能なサイクルまたは前のサイクルでキャッシュをミスした(そして、まだ再試行を行なっていない)オペランドにアクセスするサイクルでアーリ・ターミネーションが発生しないようにしています。この例で、(C)は次のすべてのサイクルのアーリ・ターミネーションを防止します。つまり、すべてのCPU空間アクセス、すべてのライト・サイクル(ライト・スルー・キャッシュが実装されているものと仮定)、$\overline{CIOUT}$がアサートされたサイクル、および前のサイクルでキャッシュ・ミスを起こしたサイクルのうちキャッシュ不可能ロケーションへのアクセスではなかったものです。(C)のフリップ・フロップは$\overline{AS}$の立上りエッジで、現在のバス・サイクルの終了条件をラッチし、この条件を次のサイクルで使用します。システムで要求があれば、アーリ・ターミネーションを抑止するための別の条件を含めることができますが、ステートS1の立上りエッジの前で(C) の出

力が有効になるように、伝搬遅延には十分配慮しておく必要があります(等式12-3を参照)。

レイト・ターミネーション回路はゲート(D)とゲート(E)で構成されます。現在のサイクルが、(C)の出力で決められたキャッシュ可能なロケーションをアクセスしていて、キャッシュ・ヒットが発生しなかった場合(D)、$\overline{\text{BERR}}$および$\overline{\text{HALT}}$信号が"L"にドライブされます(E)。

なお、図12-18に示すロジックは、ノー・ウエイト・ステートで動作するキャッシュをサポートするよう設計されています。(C)とMC68030の間にタイミング・ステージを追加してウエイト・ステートの発生回路を用意すれば、必要なクロック周期数だけこの出力の伝搬を遅らせることができます。

バス・サイクルの再試行において、遅延が発生する可能性を小さくするには、バス・エラーおよびホルト信号のネゲート・パスを注意深く制御しなければなりません。これらの信号ラインの容量性負荷を低減し、オープン・コレクタ・ドライバに対して適切な大きさのプルアップ抵抗を使用するか、あるいはこれと同等の方法を推奨します。

この回路の、アベイラブル・キャッシュ・タッグのルックアップ、比較、そしてロジック遅延(D)および(E)時間は、等式12-5で与えられます(20MHz のノー・ウエイト・ステートで40ns)。

その他の設計上の検討事項としては、キャッシュ・ミスを起こし再試行されるアクセスに対するメイン・メモリ・コントローラの応答があげられます。再試行操作中および論理バスに対して調停が行なわれない場合には、MC68030は再試行を通知する原因となったアドレスでアドレス・バスを継続してドライブします。そのため設計者は、この情報を活用して、メイン・メモリでアクセスを継続(または開始)することによって(最初のバス・サイクル中に$\overline{\text{AS}}$信号のステートをラッチし、再試行中にそれをアサートしたままにしておく)、そのサイクルの再試行に関連するオーバヘッドを軽減することができます。

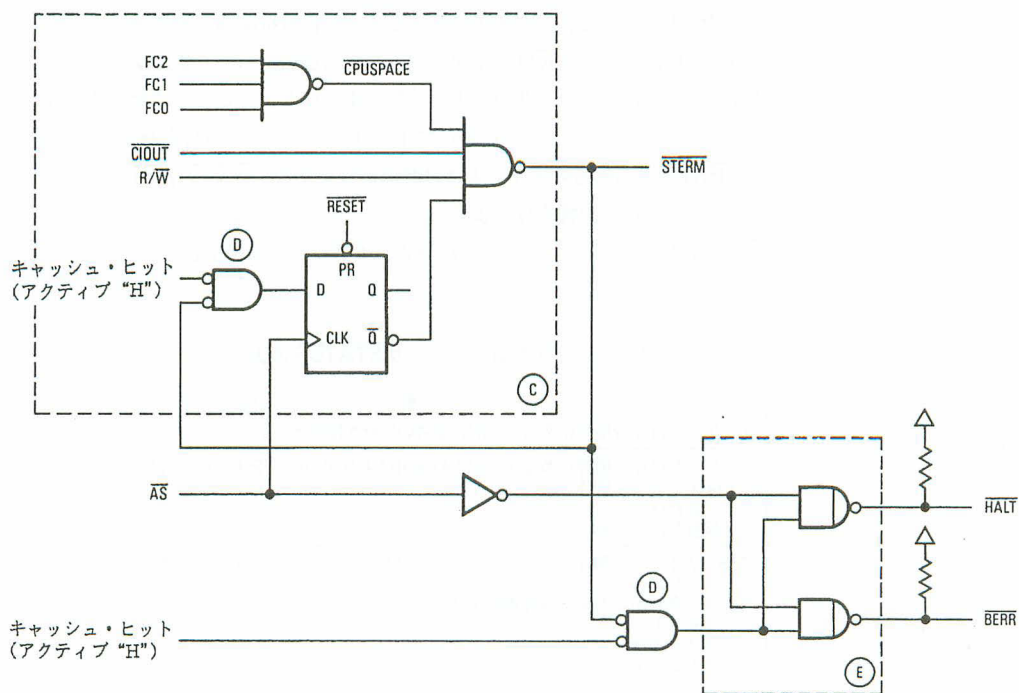



図12-18　アーリ・ターミネーション制御回路例

12. 6. 2 “命令専用”外部キャッシュのインプリメンテーション

一部のケース、特にキャッシュのコヒーレンスが重要となるマルチ・プロセッシング・システムでは、命令オペランドだけを格納することが望ましいといえます。これは命令オペランドは可変とはみなされず、ステール・データを発生しないためです。一般に、MC68000のアーキテクチャでは、PC相対アドレッシング・モードを使用しないかぎりこれを実現できます。この制限を利用すれば、ファンクション・コードをデコードすることによって、プログラムおよびデータ・アクセスを外部で区別できます。

12. 7 デバッギング・エイド

MC68030は$\overline{STATUS}$および$\overline{REFILL}$信号で、内部マイクロシーケンサのアクティビティを監視できるようになっています。これらの信号の使用法については、次の項で説明します。プログラミングのデバッギングを補助する便利なデバイスについては、「12. 7. 2 リアルタイムの命令トレース」で説明します。

12. 7. 1 $\overline{STATUS}$および$\overline{REFILL}$

MC68030は$\overline{STATUS}$および$\overline{REFILL}$信号を供給し、パイプラインのデータ処理に関係する内部マイクロシーケンサのアクティビティを識別します。バス・サイクルは、バス・コントローラで個別に制御され、スケジュールされるため、マイクロシーケンサの処理状態に関する情報は、バス信号だけを監視したのでは得られません。$\overline{STATUS}$および$\overline{REFILL}$信号で識別される内部アクティビティには、命令の境界、いくつかの例外条件、マイクロシーケンサの停止時期、および命令パイプラインの再充てんなどが含まれています。$\overline{STATUS}$と$\overline{REFILL}$は内部マイクロシーケンサのアクティビティを追跡するだけで、バス・アクティビティとは直接関係ありません。

表12-4に示すように、$\overline{STATUS}$がアサートされる連続クロック数によって、命令の境界、処理すべき例外、あるいはプロセッサの停止を示します。なお、プロセッサ停止状態は内部のエラー状態で、ダブル・バス・フォールトのためにマイクロシーケンサが自分自身を停止させているもので、外部からの$\overline{HALT}$入力信号のアサートには関係ありません。$\overline{HALT}$信号はバス操作にだけ影響し、マイクロシーケンサには影響を与えません。

$\overline{REFILL}$信号は、マイクロシーケンサが命令パイプ・ラインの再充てんを要求したことを知らせま

表12-4 マイクロシーケンサ$\overline{STATUS}$の表示

アサート期間	意　　　味
1クロック	シーケンサは命令境界にある――次の命令の実行を開始する。
2クロック	シーケンサは命令境界にあるが、以下の理由により次の命令をすぐに実行しない。 －保留トレース例外 または －保留割込み例外
3クロック	MMU アドレス変換キャッシュ・ミス――プロセッサはテーブル・サーチを開始する。 または 以下のいずれかに対する例外処理を開始する。 －リセット －バス・エラー －アドレス・エラー －スプリアス割込み －オートベクタ割込み －F系列命令(コプロセッサは応答しない)
連　続	ダブル・バス・フォールトのためにプロセッサが停止

す。再充てん要求は、非順次イベントを処理するために、命令の順次実行を中断しなければならないときに行なわれます。例外と命令の両方で$\overline{\text{REFILL}}$がアサートされます。再充てんを行なわせる命令には、分岐、ジャンプ、命令トラップ、リターン、プログラム・カウンタのフローを変更するコプロセッサの汎用命令、およびステータス・レジスタの操作などがあります。ステータス・レジスタのコンディション・コードに影響を与える論理および算術演算によっては、再充てん要求は発生しません。しかし、ステータス・レジスタを更新する“MOVE < ea >, SR”命令などの操作は、ファンクション・コードで定義されるプログラム空間を変更できるため再充てん要求が発生します。プログラム空間が変更されたときは、プロセッサは新しい空間からデータをフェッチして、前のプログラム空間からすでにプリフェッチされているデータを置き換えなければなりません。同様に、メモリ管理ユニット(MMU)のアドレス変換機構に影響を与える操作によっても再充てん要求が発生します。変換制御レジスタを変更する“PMOVE < ea >, TC”などの命令は、プロセッサに新しいアドレス変換ベースからデータをフェッチするよう要求します。“条件テスト、デクリメントおよび分岐”命令(BDcc)では、条件テストが偽のときには、2回の再充てん要求が発生します。分岐性能を最適化するために、DBcc命令は条件のテストを行なう前に再充てんを要求します。条件が偽の場合は、さらに別の再充てんを要求し、次の順次命令の実行を続けます。

図12-19にCLK信号と$\overline{\text{STATUS}}$信号で識別される通常の命令境界の関係を示します。1クロック・サイクルの間アサートされる$\overline{\text{STATUS}}$は、通常の命令境界を識別します。なお、$\overline{\text{REFILL}}$のアサートは必ずしも$\overline{\text{STATUS}}$のアサートに対応していません。$\overline{\text{STATUS}}$および$\overline{\text{REFILL}}$はともにクロック信号の立下りエッジでアサートおよびネゲートされます。

図12-20に通常の命令境界に続くトレースまたは割込み例外境界を示します。2クロック・サイクルの間アサートされる$\overline{\text{STATUS}}$によって、トレースまたは割込み例外を識別します。トレースおよび割込み例外の両方とも命令境界でのみ処理されるため、命令境界の情報が引き続き出力されてい

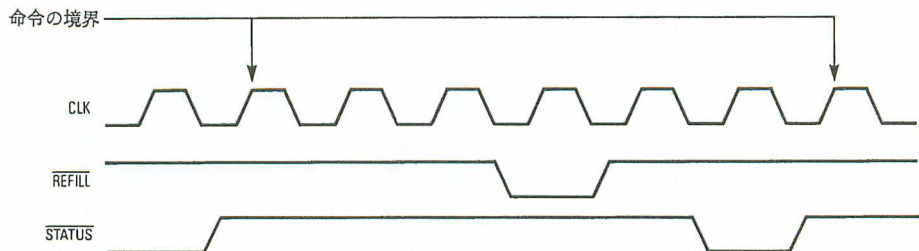



図12-19　通常の命令境界

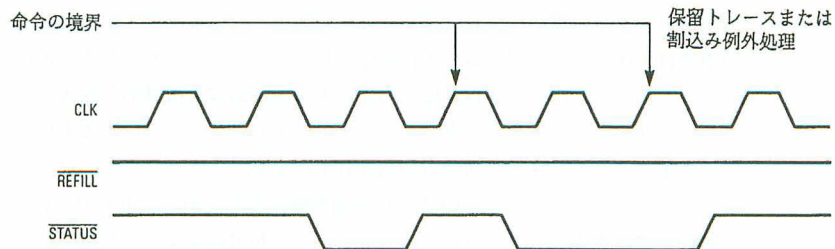



図12-20　トレースまたは割込み例外

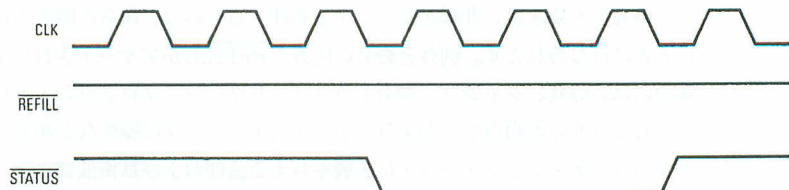



図12-21　その他の例外

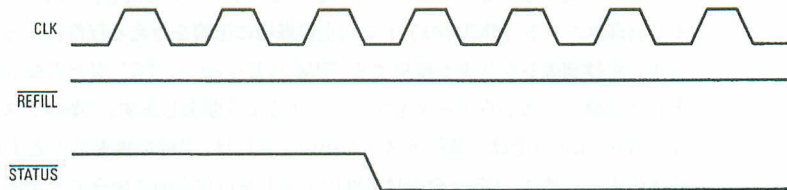



図12-22　プロセッサ停止

ます。例外ハンドラ命令がプリフェッチされる前に、$\overline{\text{REFILL}}$信号がアサートされ(図に示してない)、プログラム・フローの変更を知らせます。

図12-21に、他の例外条件に対する$\overline{\text{STATUS}}$信号のアサートを示します。これらの例外条件には、MMUアドレス変換キャッシュ・ミス、リセット、バス・エラー、アドレス・エラー、スプリアス割込み、オートベクタ割込み、コプロセッサ応答がない場合のF系列命令があります。例外処理を行なうと、3クロック・サイクルの間$\overline{\text{STATUS}}$がアサートされ、通常の命令処理が停止したことを示します。この場合、これらの例外は命令境界でない場所でも直ちに処理されるため、命令境界を決めることはできません。

図12-22にはダブル・バス・フォールトによってプロセッサが停止したことを表示する$\overline{\text{STATUS}}$のアサーションを示します。一度バス・エラーが発生したあとは、バス・エラー・ハンドラ・ルーチンの最初の命令を実行する前に、他のいかなるバス・エラー例外が発生してもダブル・バス・フォールトになります。プロセッサは、ベクタ・テーブルのリード操作中、または外部リセット後に最初の命令をプリフェッチする間にバス・エラーまたはアドレス・エラーを受け取ったときにも停止します。プロセッサを外部よりリセットするまで、$\overline{\text{STATUS}}$はアサートされたままです。

12. 7. 2　リアルタイムの命令トレース

リアルタイム・アプリケーションに使用されるマイクロプロセッサをベースにしたシステムは、一般にプログラムのデバッグを行なうための開発用ユーティリティが不足しています。リアルタイム環境では、任意にプログラムの実行を停止してデバッギングを行なうことはできません。これらのシステムには、ロボット、自動車、および工業用制御など機械的な動きを止めることができない制御アプリケーション、そしてターゲット・システムをリアルタイムで実行させたままにしておく必要のあるエミュレータ・システムが含まれています。

リアルタイム・システムにつきものの問題を解決するために、MC68030は付加ハードウェアをベースにした機能を実装して、プログラムのデバッグが可能なようにしています。リアルタイム・システムでは、リアルタイム・イベントの処理時間がなくなってしまうため、M68000プロセッサ・ファミリに組み込まれているトレース例外メカニズムを活用することはできません。MC68030には出力

ピンが追加され、リアルタイムの可視能力が与えられています。MC68030の制御信号をデコードすることによって、トレース機能を追加し、信号の追跡にどのサイクルが重要であるかを見極めることができます。また、収集したデータの事後分析によってプログラムのデバッグを行なうことができます。

外部トレース・メカニズムには、いくつかの問題があります。これらの問題には、プログラム・フローを追跡するのに、どのサイクルが重要であるかを決めること、プリフェッチ操作で取り出した命令が実行ユニットで廃棄されたかどうかを検出すること、そして外部トレース回路がオンチップ・キャッシュ・メモリへのアクセスをキャプチャ不可能か否かを検出することなどがあげられます。

プログラムのデバッグに使用する外部トレース・ハードウェアは、MC68030のバス・アクティビティに同期していなければなりません。プログラム・デバッグ環境で、すべてのクロック・サイクルをトレースするわけではないため、トレース・ハードウェアにはサンプリング信号が必要です。外部リードおよびライト操作では、データ・バスに有効データが現われると、トレース・サンプリングが発生します。2つの外部バス操作モードが可能であり、同期モードではシステムは$\overline{\text{STERM}}$信号を返し、非同期モードではシステムは$\overline{\text{DSACK1}}$または$\overline{\text{DSACK0}}$信号、あるいはそのいずれかで応答します。どちらのバス操作モードも、バスに有効なデータがあるときに、サンプリング信号を出力することが必要です。これによって、プログラムの実行を追跡する基準となるトレース・データ・フローがプロセッサに入出力できるようになります。

MC68030のパイプライン構造をもつアーキテクチャは、命令とオペランドをプリフェッチして、絶えず3段の命令パイプを充てんします。パイプラインにより、最高3ワードの単一命令または最高3つの連続命令の操作を並行して実行することができます。命令の順次実行が基準ですが、非順次イベントのためプリフェッチしたデータを実行ユニットが使用しないこともあります。$\overline{\text{STATUS}}$信号によってトレース・ハードウェアは、実行ユニットがプログラム・メモリ・オペランドを処理するときに、その実行状況を表示したり、いくつかの例外を表示することができます。非順次イベントの場合は、パイプライン全体を再ロードしてから実行を継続する必要がありますが、これは$\overline{\text{REFILL}}$信号で表示されます。

外部ハードウェアには、通常オンチップ・キャッシュ・メモリの操作は分かりません。しかし、MC68030は可視性を向上させるために、ローカル・アドレス参照を備えています。MC68030はライト・スルー方式をインプリメントして、外部ハードウェアがデータをキャプチャできるようにしているため、ライト操作は完全に見えるようになっています。オンチップ・キャッシュ・メモリからのリード操作に対しては、アドレス・バスの最下位バイトがローカル・アドレス参照を与えます。

MC68030はアドレス・バスをドライブし、外部サイクル・スタート($\overline{\text{ECS}}$)信号をアサートすることによって外部サイクルを開始します。アドレス・ストローブ($\overline{\text{AS}}$)がそのサイクルの後半でアサートされ、そのアドレスを確定します。キャッシュまたはキャッシュ保持レジスタでヒットが起こった場合は、外部サイクルがアボートされ、$\overline{\text{AS}}$はアサートされません。また、オンチップ・メモリ管理ユニット(MMU)がトレース機能に使用可能なローカル・アドレス参照を生成するために実行するアドレス変換プロセスには、下位アドレス・ビット(A0-A7)は含まれていません。オンチップ・キャッシュ・メモリからのすべてのリード・サイクルは、キャッシュ・アクセスが外部バスの可用性に依存しないため、外部でキャプチャすることはできません。

図12-23にプログラムのデバッグのためにロジック・アナライザで使用できるトレース・インタフェース回路を示します。開発中のシステムのMC68030プロセッサには、9つの入力信号($\overline{\text{DSACK1}}$、$\overline{\text{DSACK0}}$、CLK、$\overline{\text{AS}}$、$\overline{\text{RESET}}$、$\overline{\text{STATUS}}$、$\overline{\text{REFILL}}$、$\overline{\text{STERM}}$、および$\overline{\text{ECS}}$)が接続されています。また、データのキャプチャおよび分析を支援するために、6つの出力信号が生成されます。ロジック・アナライザをアドレス・バス、データ・バス、およびバス制御信号に接続するほか、トレース・インタフェース信号SAMPLE、PHALT、FILL、EP、IEおよび$\overline{\text{ECSC}}$も接続しな

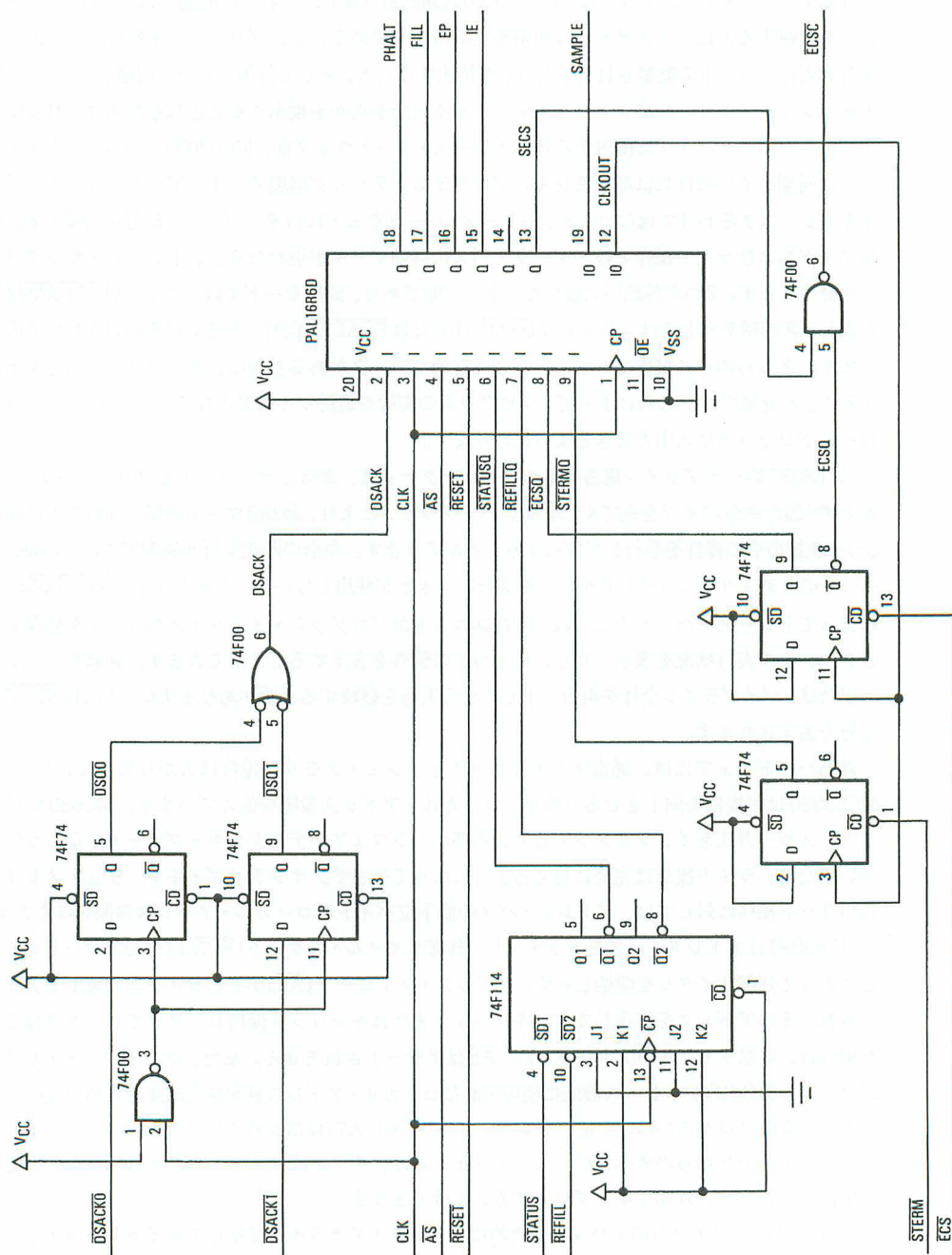



図12-23　トレース・インタフェース回路

表12-5　部品リスト

数　量	部　品	部　品　の　説　明
1	74F00	クワッド2入力NANDゲート
1	74F114	デュアルJK負エッジ・トリガ・フリップ・フロップ
2	74F74	デュアルD型正エッジ・トリガ・フリップ・フロップ
1	PAL16R6D	超高速プログラマブル・ロジック・アレイ

ければなりません。ロジック・アナライザの外部クロック・プローブは、システム CLK信号に接続して同期をとります。ロジック・アナライザをデータ・キャプチャにセットアップするときは、SAMPLE信号が“H”のときに、CLK信号の立下りエッジでサンプルを取り込む必要があります。表12-5にこの回路を実現するための部品を示します。

サンプル信号(SAMPLE)は、アクティブ“H”信号でクロック信号の次の立下りエッジをサンプリング点として認可します。次の5種類の条件により、SAMPLEがアサートされます。

1. 外部バス・サイクル
2. キャッシュ保持レジスタでのヒットを含む内部キャッシュ・ヒット
3. 命令の境界
4. 以下で説明するEP信号でマークされる例外処理
5. プロセッサの停止

残りの5つの出力信号を使用して収集した情報を認可します。

プロセッサ停止(PHALT)信号は、MC68030 がダブル・バス・フォールトを受け取り、処理を継続するにはリセット操作が必要であることを示します。PHALT は、3クロック・サイクル以上$\overline{\mathrm{STATUS}}$をアサートしたのちアサートされ、SAMPLE信号を発生します。

FILL信号は順次命令の実行の中断を示します。FILLは$\overline{\mathrm{REFILL}}$信号がラッチされたものであり、SAMPLE信号のアサートによって、サンプルの収集が通知されるまでアサートされたままになっています。FILLをアサートしてもSAMPLE信号は発生しません。

例外ペンディング(EP)信号は、MC68030がリセット、バス・エラー、アドレス・エラー、スプリアス割込み、オートベクタ割込み、F系列命令、MMUアドレス変換キャッシュ・ミス、トレース例外または割込み例外のいずれかに対する例外処理を開始していることを示します。EP信号は、$\overline{\mathrm{STATUS}}$が2〜3クロック・サイクルだけアサートされてからネゲートされたあとアサートされます。EPがアサートされると、SAMPLE信号が発生します。

命令の実行(IE)信号は、実行ユニットがちょうど1つの命令の処理を終了したことを示します。IE信号は、$\overline{\mathrm{STATUS}}$が1クロック・サイクルだけアサートされてからネゲートされたあとアサートされます。IE信号がアサートされたときにもSAMPLE信号が発生します。

外部サイクル・スタート条件($\overline{\mathrm{ECSC}}$)信号は$\overline{\mathrm{AS}}$信号と組み合わせて使用し、現在のトレース・サ

表12-6　$\overline{\mathrm{AS}}$および$\overline{\mathrm{ECSC}}$の表示

$\overline{\mathrm{AS}}$	$\overline{\mathrm{ECSC}}$	意　味
0	0	アドレスおよびデータ・バスの両方が有効
0	1	アドレスおよびデータ・バスの両方が有効
1	0	アドレス・ビット(A0-A7)が有効 アドレス・ビット(A8-A31)は無効 データ・バスは無効
1	1	アドレスおよびデータ・バスの両方が無効

```
/************************************/
/*このデバイスはMC68030の命令レベルをベースにして、プロセッサのアクティビティをトレースするための　　*
/*サンプリング信号を発生します。以下に示すピンの定義および等式では、次の記号を使用しています。　　　*
/*                          記号        定義                                                       *
/*                           !          論理NOT                                                    *
/*                           #          論理OR                                                     *
/*                           &          論理AND                                                    *
/*  さらに、'd' の拡張子の付いた信号名は、PAL の内部フリップ・フロップのD入力を示します。            *
/************************************/
/*    許容ターゲット・デバイス・タイプ：PAL16R6D    高速PAL                                          *
/************************************/
/**    入力    **/
ピン1        = clk              ;  /*ピン3のクロックと同じ          */
ピン2        = DSACK            ;  /*データ・ストローブ・アクノリッジ  */
ピン3        = CLK              ;  /* MPU クロック信号              */
ピン4        = ! AS             ;  /*アドレス・ストローブ            */
ピン5        = ! RESET          ;  /*システム・リセット信号          */
ピン6        = ! STATUSQ        ;  /*ラッチされたSTATUS信号          */
ピン7        = ! REFILLQ        ;  /*ラッチされたREFILL信号          */
ピン8        = ! ECSQ           ;  /*ラッチされたECS 信号            */
ピン9        = ! STERMQ         ;  /*ラッチされたSTERM信号           */

/**    出力    **/
ピン19       = SAMPLE           ;  /* SAMPLE信号                    */
ピン18       = PHALT            ;  /*プロセッサの停止                */
ピン17       = FILL             ;  /* REFILLの受信                   */
ピン16       = EP               ;  /*例外の保留                      */
ピン15       = IE               ;  /*命令の実行                      */
ピン14       = sc               ;  /*ステータス完了                  */
ピン13       = secs             ;  /*サンプルされたECS 信号          */
ピン12       = CLKOUT           ;  /*遅延クロック信号                */
```

図12-24　PAL のピンの定義

ンプルでアドレス・バスおよびデータ・バスが有効かどうかを判断します。表12-6に$\overline{\mathrm{AS}}$と$\overline{\mathrm{ECSC}}$の組合せを掲載し、トレースされたアドレス・バスおよびデータ・バスのどの部分が有効であるかを示します。$\overline{\mathrm{ECSC}}$がアサートされてもSAMPLE信号は発生しません。

図12-24にトレース回路に使用するPAL16R6 パッケージのピンの定義を示します。これらの定義は、図12-25に掲載するPALの等式で使用します。

12. 8　電源およびグランドの考慮事項

MC68030はモトローラの高性能HCMOSプロセスで製造され、約275,000個のトランジスタを内蔵し、最大33.33MHzのクロック周波数で動作することができます。このように多数のトランジスタを内蔵するデバイスにCMOSを使用することによって、同等のNMOS回路と比較しても大幅に電力消費を低減することができますが、デバイスを高クロック速度で動作させた場合は、供給する電源の特性が非常に重要です。電源はMC68030 がある動作を実行しているとき、大きな瞬時電流を流すことができなければならず、また、常に規定仕様範囲内になければなりません。これらの必要条件を満たすために、遅いクロック速度で動作するNMOSデバイスの場合よりも、MC68030に接続される電源に対して十分な注意を払う必要があります。

しっかりした電源インタフェースを供給するために、10本のV_{CC}ピンと14本のグランド・ピンを備えています。これによって、2本のV_{CC}ピンと4本のGNDピンでアドレス・バスに電源を供給し、2本のV_{CC}ピンと4本のGNDピンでデータ・バスに電源を供給し、さらに残りのV_{CC}および

```
/**    中間式     **/                          /*  ステート  =  PHALT  SC  EP  IE  */
S0  =  !PHALT  &  !SC  &  !EP  &  !IE;         /*    0     =    0     0   0   0   */
S1  =  !PHALT  &  !SC  &  !EP  &   IE;         /*    1     =    0     0   0   1   */
S2  =  !PHALT  &  !SC  &   EP  &   IE;         /*    2     =    0     0   1   1   */
S3  =  !PHALT  &  !SC  &   EP  &  !IE;         /*    3     =    0     0   1   0   */
S4  =   PHALT  &   SC  &   EP  &   IE;         /*    4     =    1     1   1   1   */
S5  =  !PHALT  &   SC  &  !EP  &   IE;         /*    5     =    0     1   0   1   */
S6  =  !PHALT  &   SC  &   EP  &   IE;         /*    6     =    0     1   1   1   */
S7  =  !PHALT  &   SC  &   EP  &  !IE;         /*    7     =    0     1   1   0   */

/**     論理式     **/
!SAMPLE      = !SC  &  !AS  &  !SECS  #
               !SC  &  !DSACK  &  !STERMQ  &  !SECS  #
               !SC  &   AS  &  !DSACK  &  !STERMQ  &  SECS;

!PHALT.d     = !STATUSQ  #  !EP  #  IE  #  RESET;

!SC.d        =  RESET  #
                S0  #
                S1  &   STATUSQ  #
                S2  &   STATUSQ  #
                S4  &  !STATUSQ  #
                SC  &  !PHALT;

!EP.d        =  RESET  #
                S0  #
                S1  &  !STATUSQ  #
                S4  &  !STATUSQ  #
                SC  &  !PHALT;

!IE.d        =  RESET  #
                S0  &  !STATUSQ  #
                S2  &   STATUSQ  #
                S3  &  !STATUSQ  #
                SC  &  !STATUSQ;

!SECS.d      = !ECSQ;

!CLKOUT      = !CLK;

!FILL.d      = !REFILLQ  &  SAMPLE  #
               !FILL  &  !REFILLQ  #
                RESET;
```

図12-25　論理式

表12-7　V_{CC}およびGNDピンの割当て

ピン・グループ	V_{CC}	GND
アドレス・バス	C6、D10	C5、C7、C9、E11
データ・バス	L6、K10	J11、L9、L7、L5
$\overline{ECS}$、SIZx、$\overline{DS}$、$\overline{AS}$、$\overline{DBEN}$、$\overline{CBREQ}$、R/$\overline{W}$	K4	J3
FC0-FC2、$\overline{RMC}$、$\overline{OCS}$、$\overline{CIOUT}$、$\overline{BG}$	D4	E3
内部ロジック、$\overline{RESET}$、STATUS、REFILL、その他	H3、F2、F11、H11	L8、G3、F3、G11

GNDピンを内部ロジックおよびクロック発生回路に使用することができます。表12-7にV_{CC}ピンとGNDピンの配置を掲載します。

MC68030に供給する電源のノイズを低減し、瞬時電流条件を満足するために、一般的な容量デカップリング方法を用いる必要があります。この容量デカップリングには、特に推奨レイアウトはありませんので、これらのデバイスとMC68030の間のインダクタンスを最小にして、瞬時電流要求を満たし一定の電源電圧を維持するために、十分な応答速度を与えなければなりません。それには低、中および高周波用の高品質コンデンサを組み合わせて、できるだけMC68030の近くに配置しておくとよいでしょう(たとえば、10μF、0.1μF、および330pFのコンデンサを並列に接続すれば、デジタル・システムで使用するほとんどの周波数をフィルタリングできます)。システムの他のVLSIデバイスに対しても、同様のデカップリング方法を使用してください。

電源を容量デカップリングするだけでなく、MC68030のすべてのV_{CC}およびGNDピンとシステムの電源プレーンが、低インピーダンス接続されように十分注意しなければなりません。MC68030の電源ピンとシステムの電源との接続に、十分な品質の接続を与えなかった場合は、外部信号のアサーション遅延が大きくなり、電圧のノイズ・マージンが低下するとともに、内部ロジックに電圧の誤差が生じるおそれがあります。

第 13 章

電気的特性

本章ではMC68030の電気的特性と関連のタイミングについて説明します。

13.1 最大定格

項目	記号	定格値	単位
電源電圧	V_{CC}	−0.3〜+7.0	V
入力電圧	V_{in}	−0.5〜+7.0	V
動作温度範囲	T_A	0〜70	℃
保存温度範囲	T_{stg}	−55〜150	℃

このデバイスは、各入力に対する静電気または高電界による破壊に対する保護回路を備えていますが、応用上この高インピーダンス回路に、最大定格を超えるような電圧がかからないようあらかじめ注意する必要があります。未使用ピンを適当な論理電圧(つまり、GNDまたはV_{CC})に接続しておくと、動作の信頼性が向上します。

13.2 熱特性－PGAパッケージ

項目	記号	定格値	特性値
熱抵抗－セラミック			℃/W
接合部－周囲間	θ_{JA}	30*	
接合部－ケース間	θ_{JC}	15*	

*推定値

13.3 電力条件

チップの接合部温度の平均T_J(℃)は、次式で計算できます。

$$T_J = T_A + (P_D \cdot \theta_{JA}) \quad (1)$$

ここで、

T_A＝周囲温度(℃)

θ_{JA}＝パッケージの熱抵抗、接合部－周囲間(℃/W)

$P_D = P_{INT} + P_{I/O}$

$P_{INT} = I_{CC} \times V_{CC}$(W) ── チップ内部の消費電力(W)

$P_{I/O}$＝入/出力ピンの消費電力(W) ── ユーザの使用方法による。

一般の使用条件では、$P_{I/O} < P_{INT}$であるため$P_{I/O}$は無視できます。

P_DとT_Jの関係は、次の近似式で表わされます($P_{I/O}$を無視した場合)。

$$P_D = K \div (T_J + 273℃) \quad (2)$$

(1)、(2)式からKを求めると、

$$K = P_D \cdot (T_A + 273℃) + \theta_{JA} \cdot P_D{}^2 \quad (3)$$

ここでKはデバイス固有の定数です。Kの値は既知のT_AにおけるP_D(平衡状態での値)を測定すれば、式(3)から求められます。このKの値により、(1)および(2)式から、任意のT_AにおけるP_DとT_Jを求めることができます。

パッケージの総合熱抵抗(θ_{JA})は、θ_{JC}およびθ_{CA}の2つの要素に分けることができます。ここで、θ_{JC}は接合部からパッケージ(ケース)表面までの熱流に対する抵抗を表わし、θ_{CA}はケースから周囲までの熱抵抗を表わします。これらの関係は、次式で表わされます。

$$\theta_{JA} = \theta_{JC} + \theta_{CA} \quad (4)$$

θ_{JC}はデバイスに関連する熱抵抗で、使用条件によって影響されません。しかし、θ_{CA}は使用条件に依存し、ヒート・シンク、外気の冷却、および熱伝達などの熱管理技術によって、最小限に抑えることができます。良好な熱管理のもとで使用すれば、θ_{CA}を大幅に小さくすることができ、その結果、θ_{JA}をθ_{JC}にほぼ等しくすることができます。式(1)のθ_{JA}の代わりにθ_{JC}を使用すると、半導体の接合部温度が低下します。

このデータシートに示す熱抵抗値は、推定値でないかぎり、モトローラの信頼性レポート7843「MC68XXマイクロコンポーネント・デバイスのための熱抵抗測定法」に記述されている手続きを使用して得られたもので、設計目的のためだけに記載されています。熱抵抗の測定は複雑で、手続きおよびセットアップに依存します。ユーザの測定した熱抵抗値は、データシートの値と異なる場合があります。

13. 4　DC電気的特性

(V_{CC} = 5.0Vdc ± 5 %、GND = 0Vdc、T_A = 0〜70 ℃)

項　目		記号	最小	最大	単位
"H" レベル入力電圧		V_{IH}	2.0	V_{CC}	V
"L" レベル入力電圧		V_{IL}	GND − 0.5	0.8	V
入力リーク電流 GND ≦ V_{in} ≦ V_{CC}	$\overline{\text{BERR}}$, $\overline{\text{BR}}$, $\overline{\text{BGACK}}$, CLK, $\overline{\text{IPL0-IPL2}}$, $\overline{\text{AVEC}}$, $\overline{\text{CDIS}}$, $\overline{\text{DSACK0}}$, $\overline{\text{DSACK1}}$ $\overline{\text{HALT}}$. $\overline{\text{RESET}}$	I_{in}	− 2.5 − 20	2.5 20	μ A
ハイ・インピーダンス(オフ・ステート)リーク電流 @2.4V / 0.5V	A0-A31, $\overline{\text{AS}}$, $\overline{\text{DBEN}}$, $\overline{\text{DS}}$, D0-D31, FC0-FC2, R/$\overline{\text{W}}$, $\overline{\text{RMC}}$, SIZ0-SIZ1	I_{TSI}	− 20	20	μ A
"H" レベル出力電圧 I_{OH} = 400 μA	A0-A31, $\overline{\text{AS}}$, $\overline{\text{BG}}$, D0-D31, $\overline{\text{DBEN}}$, $\overline{\text{DS}}$, $\overline{\text{ECS}}$, R/$\overline{\text{W}}$, $\overline{\text{IPEND}}$, $\overline{\text{OCS}}$, $\overline{\text{RMC}}$, SIZ0-SIZ1, FC0-FC2, $\overline{\text{CBREQ}}$, $\overline{\text{CIOUT}}$, $\overline{\text{STATUS}}$, $\overline{\text{REFILL}}$	V_{OH}	2.4	−	V
"L" レベル出力電圧 I_{OL} = 3.2 mA I_{OL} = 5.3 mA I_{OL} = 2.0 mA I_{OL} = 10.7 mA	A0-A31, FC0-FC2, SIZ0-SIZ1, $\overline{\text{BG}}$, D0-D31 $\overline{\text{CBREQ}}$, $\overline{\text{AS}}$, $\overline{\text{DS}}$, R/$\overline{\text{W}}$, $\overline{\text{RMC}}$, $\overline{\text{DBEN}}$, $\overline{\text{IPEND}}$ $\overline{\text{STATUS}}$, $\overline{\text{REFILL}}$, $\overline{\text{CIOUT}}$, $\overline{\text{ECS}}$, $\overline{\text{OCS}}$ $\overline{\text{RESET}}$	V_{OL}	− − − −	0.5 0.5 0.5 0.5	V
消費電力(TA = 0 ℃)		P_D	−	2.6	W
容量(注参照) V_{in} = 0 V, T_A = 25°C, f = 1 MHz		C_{in}	−	20	pF
負荷容量	$\overline{\text{ECS}}$, $\overline{\text{OCS}}$ $\overline{\text{CIOUT}}$, $\overline{\text{STATUS}}$, $\overline{\text{REFILL}}$ その他	C_L	−	50 70 130	pF

注：容量は100 %テストではなく、定期的にサンプリング・テストが行なわれます。

13. 5　AC電気的特性－クロック入力(図13-1参照)

番号	項　目	20MHz		25MHz		33.33MHz		単位
		最小	最大	最小	最大	最小	最大	
	動作周波数	12.5	20	12.5	25	20	33.33	MHz
1	クロックのサイクル・タイム	50	80	40	80	30	80	ns
12、3	クロックのパルス幅(1.5Vから1.5Vまで測定)	23	57	19	61	14	66	ns
4、5	クロックの立上りおよび立下り時間	－	5	－	4	－	3	ns

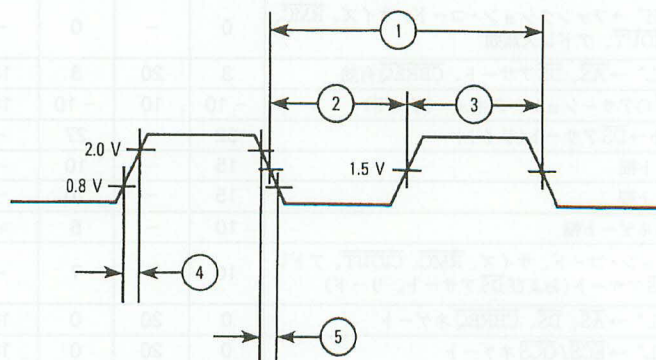



図13-1　クロック入力のタイミング図

13. 6　AC電気的特性 - リードおよびライト・サイクル

(VCC = 5.0Vdc ± 5 %、GND = 0Vdc、TA = 0〜70℃：図13-3〜13-8参照)

番号	項目	20MHz		25MHz		33.33MHz		単位
		最小	最大	最小	最大	最小	最大	
6	クロック "H" →ファンクション・コード、サイズ、$\overline{RMC}$、$\overline{IPEND}$、$\overline{CIOUT}$、アドレス有効	0	25	0	20	0	14	ns
6A	クロック "H" →$\overline{ECS}$、$\overline{OCS}$アサート	0	15	0	15	0	12	ns
6B	ファンクション・コード、サイズ、$\overline{RMC}$、$\overline{IPEND}$、$\overline{CIOUT}$、アドレス有効→$\overline{ECS}$のネゲート・エッジ	4	−	3	−	3	−	ns
7	クロック "H" →ファンクション・コード、サイズ、$\overline{RMC}$、$\overline{CIOUT}$、アドレス、データ・ハイ・インピーダンス	0	50	0	40	0	30	ns
8	クロック "H" →ファンクション・コード、サイズ、$\overline{RMC}$、$\overline{IPEND}$、$\overline{CIOUT}$、アドレス無効	0	−	0	−	0	−	ns
9	クロック "L" →$\overline{AS}$、$\overline{DS}$アサート、$\overline{CBREQ}$有効	3	20	3	18	2	10	ns
9A[1]	$\overline{AS}$から$\overline{DS}$のアサーション・スキュー(リード)	−10	10	−10	10	−8	8	ns
9B[14]	$\overline{AS}$アサート→$\overline{DS}$アサート(ライト)	32	−	27	−	22	−	ns
10	$\overline{ECS}$アサート幅	15	−	10	−	8	−	ns
10A	$\overline{OCS}$アサート幅	15	−	10	−	8	−	ns
10B[7]	$\overline{ECS}$、$\overline{OCS}$ネゲート幅	10	−	5	−	5	−	ns
11	ファンクション・コード、サイズ、$\overline{RMC}$、$\overline{CIOUT}$、アドレス有効→$\overline{AS}$アサート(および$\overline{DS}$アサート、リード)	10	−	7	−	5	−	ns
12	クロック "L" →$\overline{AS}$、$\overline{DS}$、$\overline{CBREQ}$ネゲート	0	20	0	18	0	10	ns
12A	クロック "L" →$\overline{ECS}$/$\overline{OCS}$ネゲート	0	20	0	18	0	15	ns
13	$\overline{AS}$、$\overline{DS}$ネゲート→ファンクション・コード、サイズ、$\overline{RMC}$、$\overline{CIOUT}$、アドレス無効	10	−	7	−	5	−	ns
14	$\overline{AS}$(および$\overline{DS}$リード)アサート幅(非同期サイクル)	85	−	70	−	45	−	ns
14A[11]	$\overline{DS}$アサート幅(ライト)	38	−	30	−	23	−	ns
14B	$\overline{AS}$(および$\overline{DS}$リード)アサート幅(同期サイクル)	35	−	30	−	23	−	ns
15	$\overline{AS}$、$\overline{DS}$ネゲート幅	38	−	30	−	23	−	ns
15A[8]	$\overline{DS}$ネゲート→$\overline{AS}$アサート	30	−	25	−	18	−	ns
16	ｸﾛｯｸ"H"→$\overline{AS}$、$\overline{DS}$、R/$\overline{W}$、$\overline{DBEN}$、$\overline{CBREQ}$ﾊｲ・ｲﾝﾋﾟｰﾀﾞﾝｽ	−	50	−	40	−	30	ns
17	$\overline{AS}$、$\overline{DS}$ネゲート→R/$\overline{W}$無効	10	−	7	−	5	−	ns
18	クロック "H" →R/$\overline{W}$ "H"	0	25	0	20	0	15	ns
20	クロック "H" →R/$\overline{W}$ "L"	0	25	0	20	0	15	ns
21	R/$\overline{W}$ "H" →$\overline{AS}$アサート	10	−	7	−	5	−	ns
22	R/$\overline{W}$ "L" →$\overline{DS}$アサート(ライト)	60	−	47	−	35	−	ns
23	クロック "H" →データ出力有効	−	25	−	20	−	14	ns
24	データ出力有効→$\overline{AS}$のネゲート・エッジ	8	−	5	−	3	−	ns
25[11]	$\overline{AS}$、$\overline{DS}$ネゲート→データ出力無効	10	−	7	−	5	−	ns
25A[9,11]	$\overline{DS}$ネゲート→$\overline{DBEN}$ネゲート(ライト)	10	−	7	−	5	−	ns
26[11]	データ出力有効→$\overline{DS}$アサート(ライト)	10	−	7	−	5	−	ns
27	データ入力有効→クロック "L"(セットアップ)	4	−	2	−	1	−	
27A	ﾚｲﾄ$\overline{BERR}$/$\overline{HALT}$ｱｻｰﾄ→ｸﾛｯｸ"L"(ｾｯﾄｱｯﾌﾟ)	10	−	5	−	3	−	ns
28[12]	$\overline{AS}$、$\overline{DS}$ネゲート→$\overline{DSACKx}$、$\overline{BERR}$、$\overline{HALT}$、$\overline{AVEC}$ネゲート(非同期ホールド)	0	50	0	40	0	30	ns
28A[12]	クロック "L" →$\overline{DSACKx}$、$\overline{BERR}$、$\overline{HALT}$、$\overline{AVEC}$ネゲート(同期ホールド)	12	85	8	70	6	50	ns
29[12]	$\overline{AS}$、$\overline{DS}$ネゲート→データ入力無効(非同期ホールド)	0	−	0	−	0	−	ns

AC電気的特性(つづき)

番号	項目	20MHz		25MHz		33.33MHz		単位
		最小	最大	最小	最大	最小	最大	
29A[12]	$\overline{AS}$、$\overline{DS}$ネゲート→データ入力ハイ・インピーダンス	–	50	–	40	–	30	ns
30[12]	クロック"L"→データ入力無効(同期ホールド)	12	–	8	–	6	–	ns
30A[12]	クロック"L"→データ入力ハイ・インピーダンス(リードの後ライト)	–	75	–	60	–	45	ns
31[2]	$\overline{DSACKx}$アサート→データ入力有効(非同期データ・セットアップ)	–	43	–	28	–	20	ns
31A[3]	$\overline{DSACKx}$アサート→$\overline{DSACKx}$有効(スキュー)	–	10	–	7	–	5	ns
32	$\overline{RESET}$入力遷移時間	–	1.5	–	1.5	–	1.5	Clks
33	クロック"L"→$\overline{BG}$アサート	0	25	0	20	0	15	ns
34	クロック"L"→$\overline{BG}$ネゲート	0	25	0	20	0	15	ns
35	$\overline{BR}$アサート→$\overline{BG}$アサート($\overline{RMC}$は非アサート時)	1.5	3.5	1.5	3.5	1.5	3.5	Clks
37	$\overline{BGACK}$アサート→$\overline{BG}$ネゲート	1.5	3.5	1.5	3.5	1.5	3.5	Clks
37A	$\overline{BGACK}$アサート→$\overline{BR}$ネゲート	0	1.5	0	1.5	0	1.5	Clks
39[6]	$\overline{BG}$ネゲート幅	75	–	60	–	45	–	ns
39A	$\overline{BG}$アサート幅	75	–	60	–	45	–	ns
40	クロック"H"→$\overline{DBEN}$アサート(リード)	0	25	0	20	0	18	ns
41	クロック"L"→$\overline{DBEN}$ネゲート(リード)	0	25	0	20	0	18	ns
42	クロック"L"→$\overline{DBEN}$アサート(ライト)	0	25	0	20	0	18	ns
43	クロック"H"→$\overline{DBEN}$ネゲート(ライト)	0	25	0	20	0	18	ns
44	R/$\overline{W}$"L"→$\overline{DBEN}$アサート(ライト)	10	–	7	–	5	–	ns
45[5]	$\overline{DBEN}$アサート幅 非同期リード 非同期ライト	50 100	– –	40 80	– –	30 60	– –	ns
45A[9]	$\overline{DBEN}$アサート幅 同期リード 同期ライト	10 50	– –	5 40	– –	5 30	– –	ns
46	R/$\overline{W}$アサート幅(非同期ライトまたはリード)	125	–	100	–	75	–	ns
46A	R/$\overline{W}$アサート幅(同期ライトまたはリード)	75	–	60	–	45	–	ns
47A	非同期入力セットアップ時間→クロック"L"	4	–	2	–	2	–	ns
47B	クロック"L"→非同期入力ホールド時間	12	–	8	–	6	–	ns
48[4]	$\overline{DSACKx}$アサート→$\overline{BERR}$、$\overline{HALT}$アサート	–	20	–	25	–	18	ns
53	クロック"H"からのデータ出力ホールド時間	3	–	3	–	2	–	ns
55	R/$\overline{W}$アサート→データ・バスのインピーダンス変化	25	–	20	–	15	–	ns
56	$\overline{RESET}$パルス幅(リセット命令)	512	–	512	–	512	–	Clks
57	$\overline{BERR}$ネゲート→$\overline{HALT}$ネゲート(再実行)	0	–	0	–	0	–	ns
58[10]	$\overline{BGACK}$ネゲート→バス駆動	1	–	1	–	1	–	Clks
59[10]	$\overline{BG}$ネゲート→バス駆動	1	–	1	–	1	–	Clks
60[13]	同期入力有効→クロック"H"(セットアップ時間)	4	–	2	–	2	–	ns
61[13]	クロック"H"→同期入力無効(ホールド時間)	12	–	8	–	6	–	ns
62	クロック"L"→$\overline{STATUS}$、$\overline{REFILL}$アサート	0	25	0	20	0	15	ns
63	クロック"L"→$\overline{STATUS}$、$\overline{REFILL}$ネゲート	0	25	0	20	0	15	ns

注：1. ストローブの負荷が等しい場合、この値は5nsに減らすことができます。

2. 非同期のセットアップ・タイム(#47A)の条件が満足されている場合、$\overline{DSACKx}$"L"に対するデータのセットアップ・タイム(#31)および$\overline{DSACKx}$"L"に対する$\overline{BERR}$の"L"のセットアップ・タイム(#48)は無視することができます。その場合、データはクロックの"L"に対するデータ入力のセットアップ・タイム(#27)を、次のクロック・サイクルに対して満足すればよく、$\overline{BERR}$はクロック"L"に対するレイト$\overline{BERR}$"L"のセットアップ・タイムを次のクロックに対して満足するだけですみます。

3. このパラメータは、$\overline{DSACK0}$アサートから$\overline{DSACK1}$アサート、または$\overline{DSACK1}$アサートから$\overline{DSACK0}$アサート間のスキューの最大許容値を規定しています。$\overline{DSACK0}$または$\overline{DSACK1}$はパラメータ#47Aを満足していなければなりません。

4. この仕様は最初にアサートされる$\overline{DSACKx}$信号($\overline{DSACK0}$または$\overline{DSACK1}$)に適用されます。$\overline{DSACKx}$がアサートされなかった場合、$\overline{BERR}$は非同期入力のセットアップ・タイム(#47A)が適用される非同期入力です。

5. $\overline{DBEN}$はライト・サイクルが続いている間、アサートされたままでもかまいません。

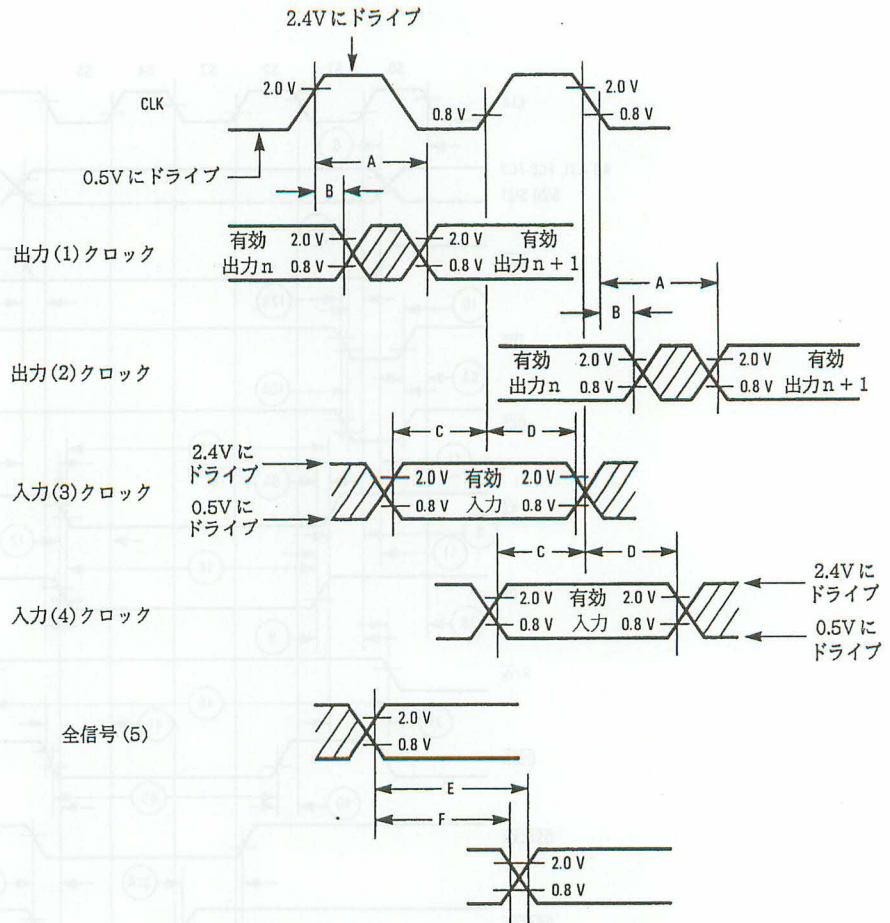
6. 正しい動作を保証するために、最小値を満足しなければなりません。最大値が満足されなかった場合は、$\overline{\mathrm{BG}}$は再度アサートされます。
7. この仕様は内部キャッシュ・ヒットの直後に、別のキャッシュ・ヒット、キャッシュ・ミス、またはオペランド・サイクルが発生したときの$\overline{\mathrm{ECS}}$および$\overline{\mathrm{OCS}}$の最小"H"時間を示します。
8. この仕様はMC68881/MC68882との動作を保証するもので、$\overline{\mathrm{DS}}$がネゲートされてから$\overline{\mathrm{AS}}$がアサートされるまでの最小時間を規定します(MC68881/MC68882ユーザーズ・マニュアルの仕様#13A)。この仕様がない場合、仕様#9Aおよび#15がMC68030がMC68881/MC68882の条件に適合しないというふうに、誤って解釈されてしまいます。
9. この仕様により、システム設計者は、$\overline{\mathrm{DBEN}}$で生成される出力イネーブル信号をもつデータ・バッファの出力側のデータ・ホールド時間を保証することができます。$\overline{\mathrm{DBEN}}$のタイミングのために、ノー・ウエイト・ステートの同期リード・サイクルに使用することはできません。
10. これらの仕様により、システム設計者は、MC68030が調停シーケンスの後で、バスの制御権を取り戻したときに、別のバス・マスタがバスのドライブを停止するよう保証することができます。
11. ノー・ウエイト・ステートの同期ライト・サイクルでは、$\overline{\mathrm{DS}}$はアサートされません。
12. これらのホールド時間は、ストローブ(非同期)およびクロック(同期)に対して規定されます。どちらを使用するかは設計者の自由です。
13. 同期入力は、$\overline{\mathrm{AS}}$がアサートされている間は、クロックのすべての立上りエッジに対して、安定したロジック・レベルをもち、仕様#60および#61を満足しなければなりません。これらの値は立上りクロック・エッジの"H"レベルを基準にして規定されています。従来、公表されていた値はクロック・エッジの"L"レベルを基準にして規定されていました。
14. この仕様により、システム設計者はMC68881/MC68882の$\overline{\mathrm{CS}}$信号を$\overline{\mathrm{AS}}$で認知しながら(ゲート遅延に7nsを許容)、まだMC68881/MC68882の$\overline{\mathrm{CS}}$に対する$\overline{\mathrm{DS}}$のセットアップ時間条件(仕様8B)を満たすことができます。

タイミング図を図13-3から13-8に示します。

13.7 AC電気的仕様の定義

本書に記載するAC仕様は、出力遅延時間、入力セットアップおよびホールド時間、そして信号スキュー時間よりなります。すべての信号は、MC68030のクロック入力、およびそれ以外の1つまたは複数の信号の適切なエッジを基準にして規定されています。

AC仕様の測定は、図13-2の波形で定義されます。モトローラで保証するパラメータをテストするために、入力は図13-2で指定される電圧レベルにドライブしなければなりません。MC68030の出力は最小および最大制限値、あるいはそのいずれか該当する値で、図に示すとおり測定されます。MC68030への入力は最小(および該当する場合は、最大)セットアップおよびホールド時間で規定されており、図に示すとおり測定されます。最後に、信号-信号間の仕様の測定も図に示してあります。

なお、MC68030のAC仕様への適合を確認するために使用するテスト・レベルは、DC電気的仕様で規定されるデバイスの保証DC動作に影響を与えません。

注：1.　この出力タイミングはクロックの立上りエッジを基準にして規定される全パラメータに適用されます。
2.　この出力タイミングはクロックの立下りエッジを基準にして規定される全パラメータに適用されます。
3.　この入力タイミングはクロックの立上りエッジを基準にして規定される全パラメータに適用されます。
4.　この入力タイミングはクロックの立下りエッジを基準にして規定される全パラメータに適用されます。
5.　このタイミングは別の信号のアサーション/ネゲーションを基準にして規定される全パラメータに適用されます。

凡例：A－最大出力遅延仕様
B－最小出力ホールド時間
C－最小入力セットアップ時間仕様
D－最小入力ホールド時間仕様
E－信号有効から信号有効までの仕様(最大または最小)
F－信号有効から信号無効までの仕様(最大または最小)

図13-2　AC仕様のドライブ・レベルとテスト・ポイント

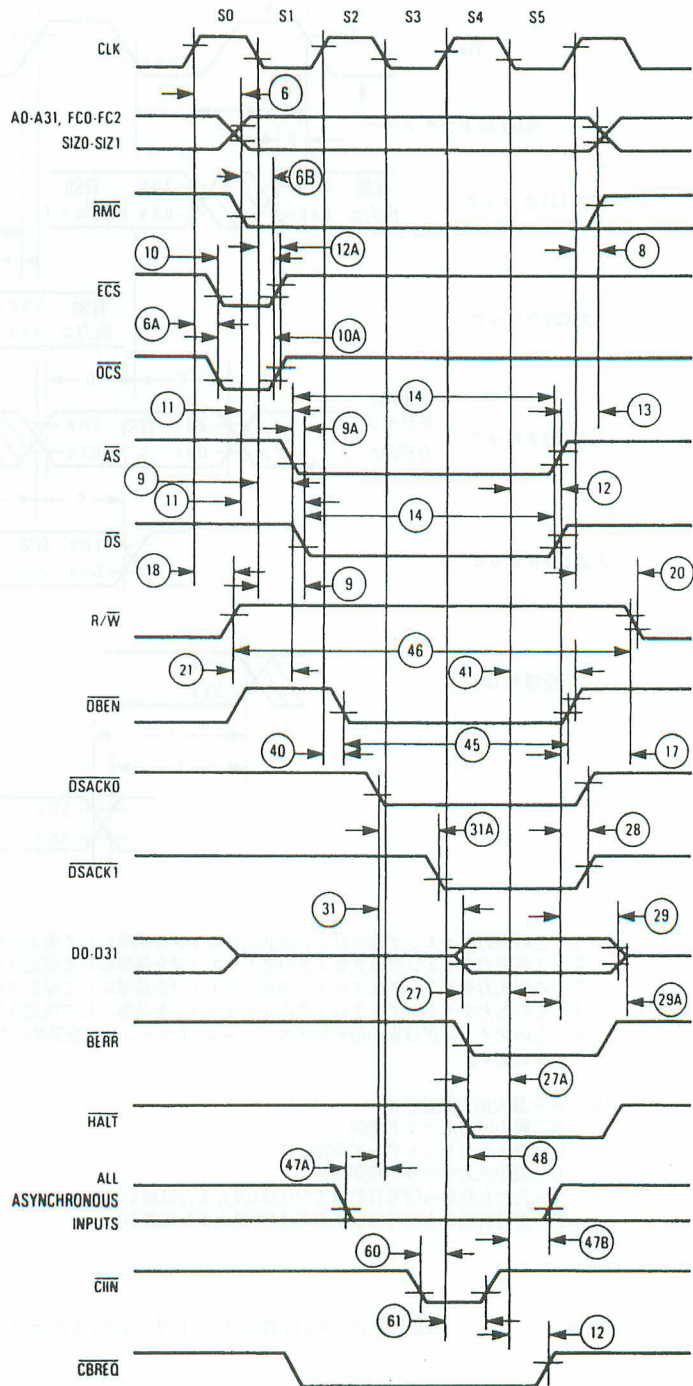



図13-3　非同期リード・サイクルのタイミング図

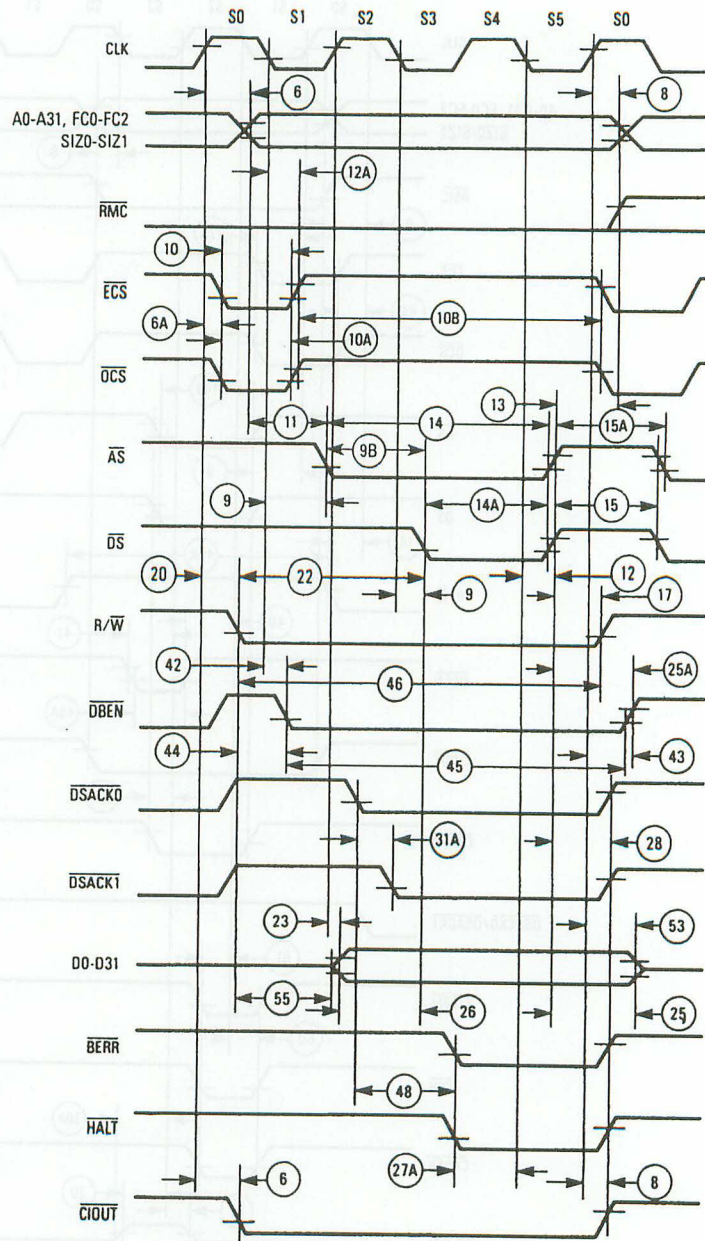



図13-4　非同期ライト・サイクルのタイミング図

図13-5　同期リード・サイクルのタイミング図

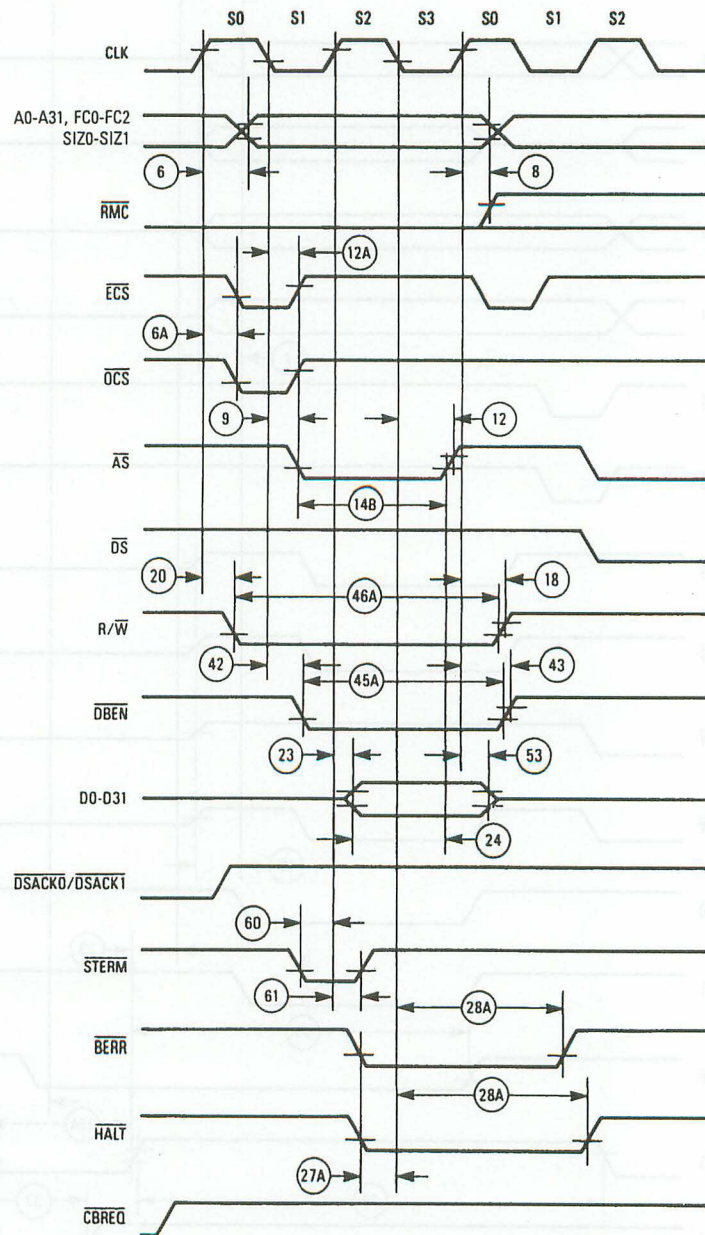



図13-6　同期ライト・サイクルのタイミング図

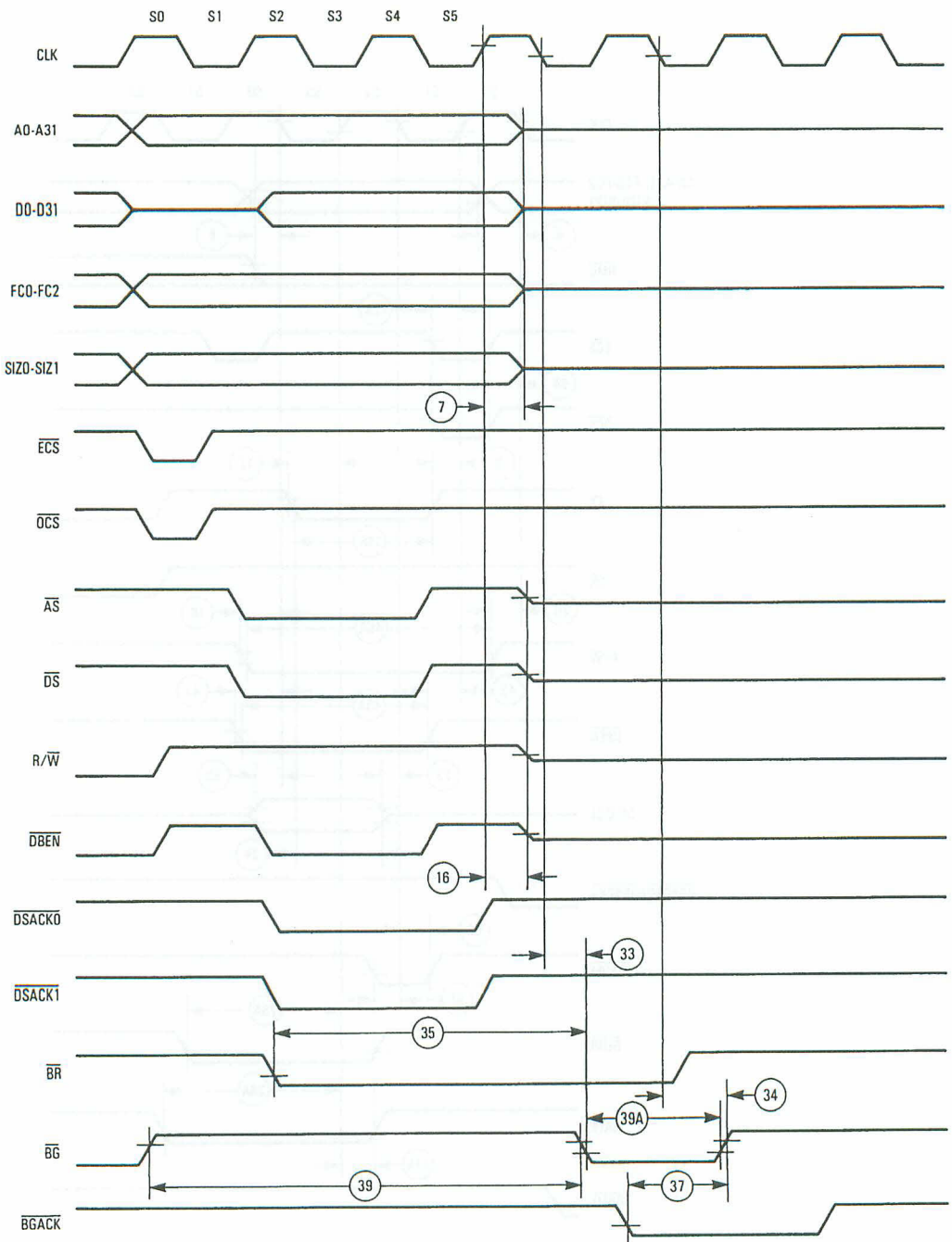



図13-7　バス調停のタイミング図

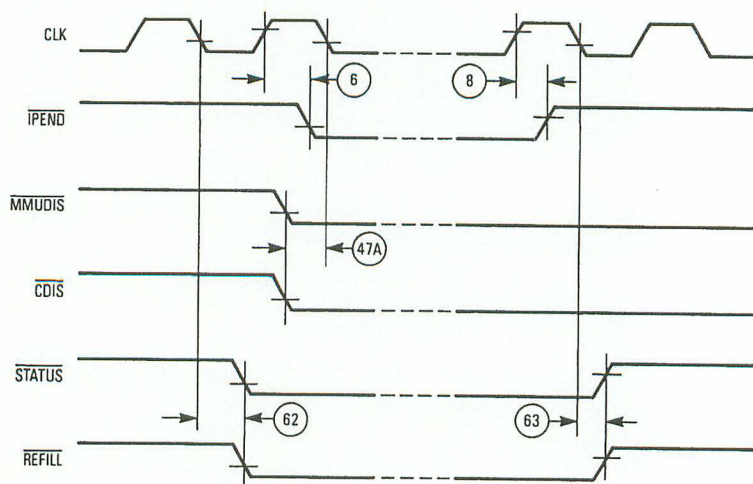



図13-8　他の信号のタイミング図

第14章

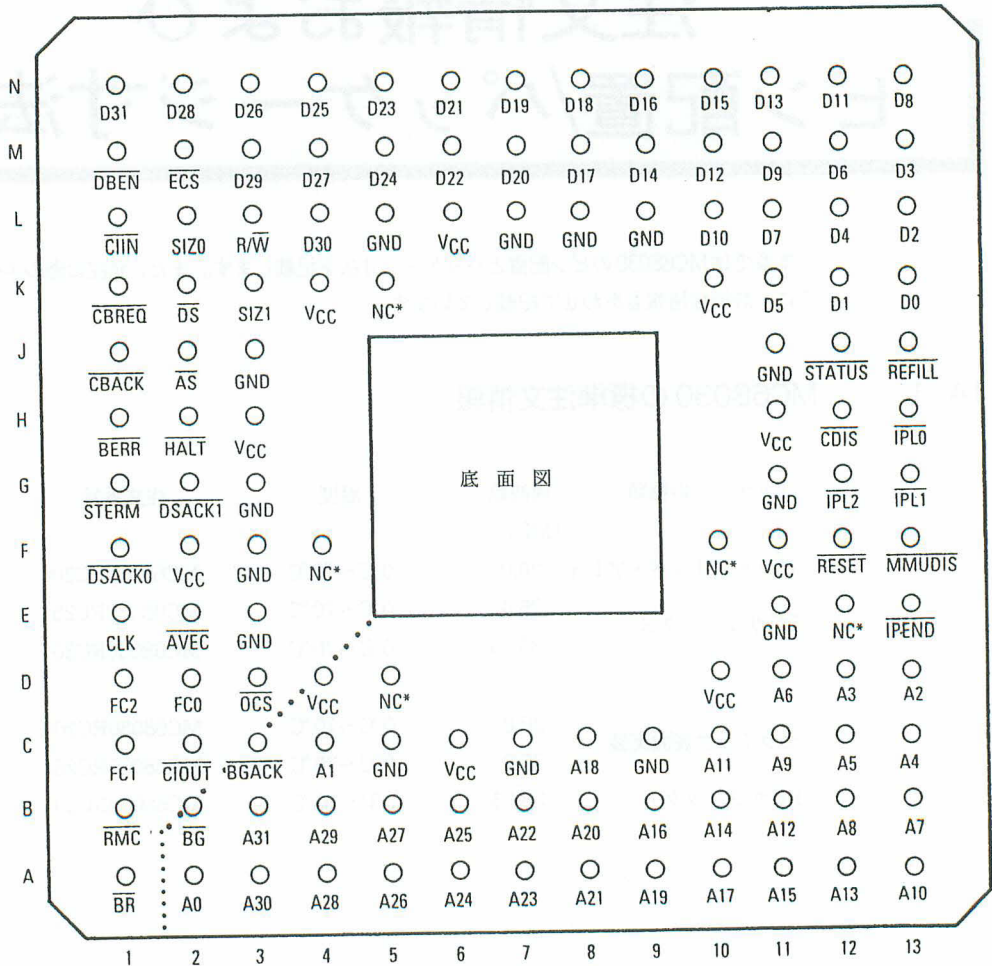
注文情報および
ピン配置/パッケージ寸法

本章ではMC68030のピン配置とパッケージ寸法を記載します。また、発注の際の手引きになるように、詳細な情報もあわせて記載しています。

14. 1　MC68030の標準注文情報

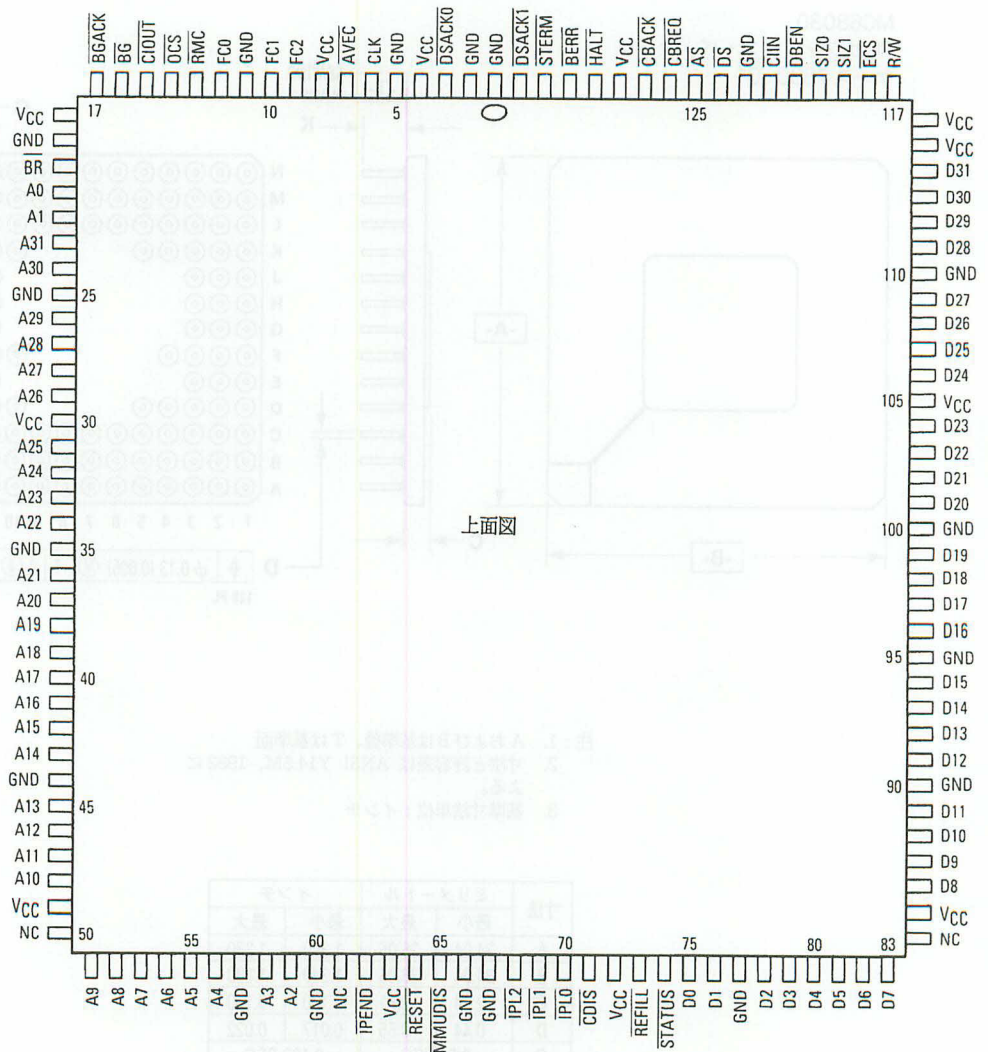
パッケージの種類	周波数 (MHz)	温度	注文番号
ピン・グリッド・アレイ	20.0	0℃~70℃	MC68030RC20
RCサフィックス	25.0	0℃~70℃	MC68030RC25
	33.33	0℃~70℃	MC68030RC30
セラミック表面実装	20.0	0℃~70℃	MC68030RC20
FEサフィックス	25.0	0℃~70℃	MC68030RC25
	33.33	0℃~70℃	MC68030RC30

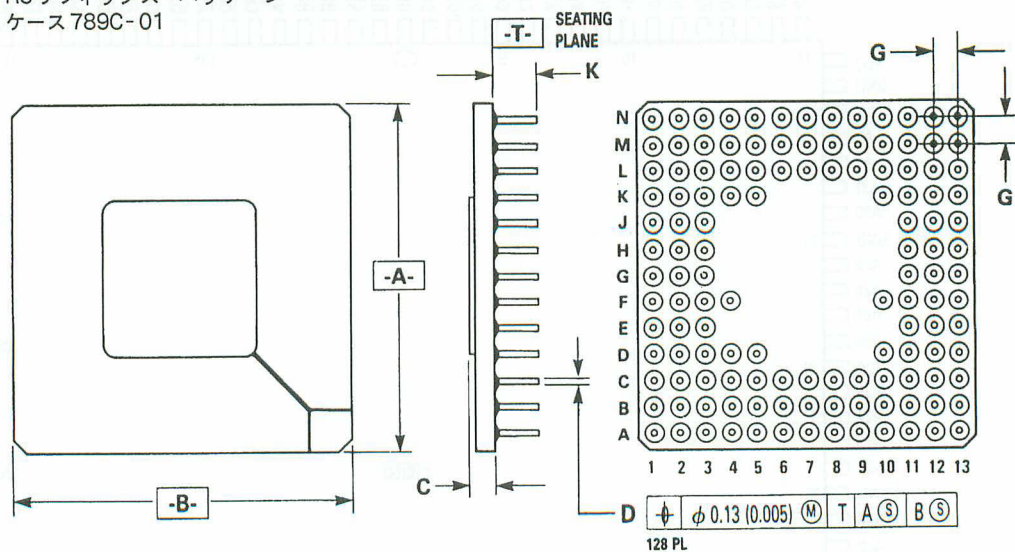
14.2 ピン配置－ピン・グリッド・アレイ（RCサフィックス）

V_{CC}およびGNDピンはアドレス・バス・バッファ、データ・バス・バッファ、そして他のすべての出力バッファおよび内部ロジックに対して、個別に電源を供給するために、3つのグループに分類されています。

＊NC—このピンには接続しないでください。

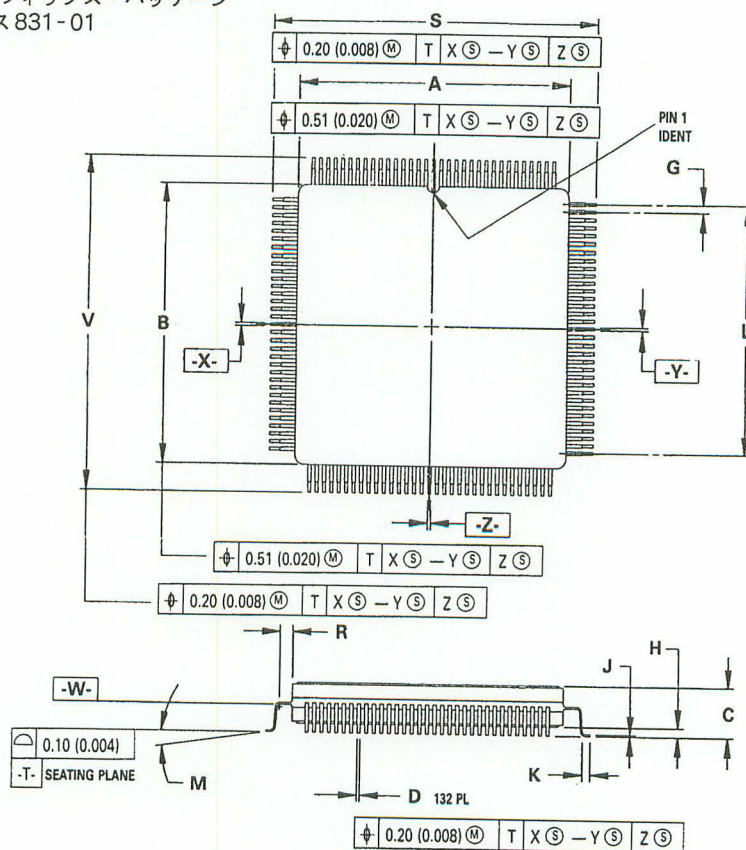
ピン・グループ	V_{CC}	GND
アドレス・バス	C6, D10	C5, C7, C9, E11
データ・バス	L6, K10	J11, L9, L7, L5
$\overline{\text{ECS}}$, SIZx, $\overline{\text{DS}}$, $\overline{\text{AS}}$, $\overline{\text{DBEN}}$, $\overline{\text{CBREQ}}$, R/$\overline{\text{W}}$	K4	J3
FC0-FC2, $\overline{\text{RMC}}$, $\overline{\text{OCS}}$, $\overline{\text{CIOUT}}$, $\overline{\text{BG}}$	D4	E3
内部ロジック，$\overline{\text{RESET}}$, STATUS, REFILL, その他	H3, F2, F11, H11	L8, G3, F3, G11

14.3　ピン配置 - セラミック・サーフェス・マウント(FEサフィックス)

＊NC－このピンには接続しないでください。

14. 4 パッケージ寸法図

MC68030
RCサフィックス・パッケージ
ケース789C-01

注：1. AおよびBは基準値、Tは基準面
2. 寸法と許容差は ANSI Y14.5M、1982 による。
3. 基準寸法単位：インチ

寸法	ミリメートル		インチ	
	最小	最大	最小	最大
A	34.04	35.05	1.340	1.380
B	34.04	35.05	1.340	1.380
C	2.54	3.81	0.100	0.150
D	0.44	0.55	0.017	0.022
G	2.54 BSC		0.100 BSC	
K	4.32	4.95	0.170	0.195

MC68030
FEサフィックス・パッケージ
ケース831-01

寸法	ミリメートル		インチ	
	最小	最大	最小	最大
A	21.85	22.86	0.860	0.900
B	21.85	22.86	0.860	0.900
C	3.94	4.31	0.155	0.170
D	0.204	0.292	0.0080	0.0115
G	0.64 BSC		0.025 BSC	
H	0.64	0.88	0.025	0.035
J	0.13	0.20	0.005	0.008
K	0.51	0.76	0.020	0.030
L	20.32 REF		0.800 REF	
M	0°	8°	0°	8°
R	0.64	—	0.025	—
S	27.31	27.55	1.075	1.085
V	27.31	27.55	1.075	1.085

注：1. 寸法と許容差は ANSI Y14.5M、1982による。
2. 基準寸法単位：インチ
3. 寸法AおよびBはガラス繊維の突起およびセラミック・ボディの上部と下部の不整合を含む、最大セラミック・ボディ寸法を定義する。
4. 基準面-W-はリードがパッケージのボディから出た位置のリードの下側です。
5. X－YおよびZは基準面-W-において、中央のリードがパッケージのボディから出た位置において決定される。
6. 寸法SおよびVは設置面-T-において決定される。
7. 寸法AおよびBは基準面-W-において決定される。

付録A

この付録Aは、M68000ファミリ・マイクロプロセッサの特長をまとめたものです。*M68000 Programmer's Reference Manual*には、MC68000とMC68010の違いがより詳細に記載されています。

	MC68000	MC68008	MC68010	MC68020	MC68030
Data Bus Size (Bits)	16	8	16	8,16,32	8,16,32
Address Bus Size (Bits)	24	20	24	32	32
Instruction Cache (in words)	—	—	3[1]	128	128
Data Cache (in words)	—	—	—	—	128

Note 1. The MC68010 supports a 3-word cache for the loop mode.

Virtual Memory/Machine

MC68010, MC68020, and MC68030	Provide Bus Error Detection, Fault Recovery
MC68030	On-chip MMU

Coprocessor Interface

MC68000, MC68008, and MC68010	Emulated in software
MC68020 and MC68030	In Microcode

Word/Long Word Data Alignment

MC68000, MC68008, and MC68010	Word/Long Data, Instructions, and Stack Must be Word Aligned
MC68020 and MC68030	Only Instructions Must be Word Aligned (Data Alignment Improves Performance)

Control Registers

MC68000 and MC68008	None
MC68010	SFC, DFC, VBR
MC68020	SFC, DFC, VBR, CACR, CAAR
MC68030	SFC, DFC, VBR, CACR, CAAR, CRP, SRP, TC, TT0, TT1, PSR

Stack Pointers

MC68000, MC68008, and MC68010	USP, SSP
MC68020 and MC68030	USP, SSP (MSP, ISP)

Status Register Bits

MC68000, MC68008, and MC68010	T, S, I0/I1/I2, X/N/Z/V/C
MC68020 and MC68030	T0/T1, S, M, I0/I1/I2, X/N/Z/V/C

Function Code/Address Space

MC68000 and MC68008	FC0-FC2=7 is Interrupt Acknowledge, Only
MC68010, MC68020, and MC68030	FC0-FC2=7 is CPU Space

Indivisible Bus Cycles

MC68000, MC68008, and MC68010	Use $\overline{\text{AS}}$ Signal
MC68020 and MC68030	Use $\overline{\text{RMC}}$ Signal

Stack Frames

MC68000 and MC68008	Support Original Set
MC68010	Supports Formats $0, $8
MC68020 and MC68030	Support Formats $0, $1, $2, $9, $A, $B

Addressing Modes

MC68020 and MC68030 extensions:	Memory indirect addressing modes, scaled index, and larger displacements. Refer to specific data sheets for details.

MC68020 and MC68030 Instruction Set Extensions

Bcc	Supports 32-Bit Displacements
BFxxxx	Bit Field Instructions (BFCHG, BFCLR, BFEXTS, BFEXTU, BFFFO, BFINS, BFSET, BFTST)
BKPT	New Instruction Functionality
BRA	Supports 32-Bit Displacements
BSR	Supports 32-Bit Displacements
CALLM	New Instruction (MC68020 only)
CAS, CAS2	New Instructions
CHK	Supports 32-Bit Operands
CHK2	New Instruction
CMPI	Supports Program Counter Relative Addressing Modes
CMP2	New Instruction
cp	Coprocessor Instructions
DIVS/DIVU	Supports 32-Bit and 64-Bit Operands
EXTB	Supports 8-Bit Extend to 32 Bits
LINK	Supports 32-Bit Displacement
MOVEC	Supports New Control Registers
MULS/MULU	Supports 32-Bit Operands
PACK	New Instruction
PFLUSH	MMU Instruction (MC68030 only)
PLOAD	MMU Instruction (MC68030 only)
PMOVE	MMU Instruction (MC68030 only)
PTEST	MMU Instruction (MC68030 only)
RTM	New Instruction (MC68020 only)
TST	Supports Program Counter Relative Addressing Modes
TRAPcc	New Instruction
UNPK	New Instruction

MC68030ユーザーズ・マニュアル

1990年12月20日　第1版第1刷発行

1993年7月20日　第1版第4刷発行

原文 MC68030 ENHANCED 32-BIT MICROPROCESSOR USER'S MANUAL SECOND EDITION (MC68030 UM/AD REV1) by Motorola Inc.

日本モトローラ株式会社　半導体事業部

〒106 東京都港区南麻布3-20-1

03-3440-3311

MC68030UMJ/AD

NML:AD93080-MPU16-RN JUL/'93 SPEC 1K

MC68030ユーザーズマニュアル（和文）

定価2,575円
（本体2,500円）

発行所　日本モトローラ株式会社
〒106　東京都港区南麻布5-2-32
☎03(3440)3311（大代表）

発売元　ＣＱ出版株式会社
〒170　東京都豊島区巣鴨1-14-2
☎03(5395)2141（営業）　振替　東京0-10665